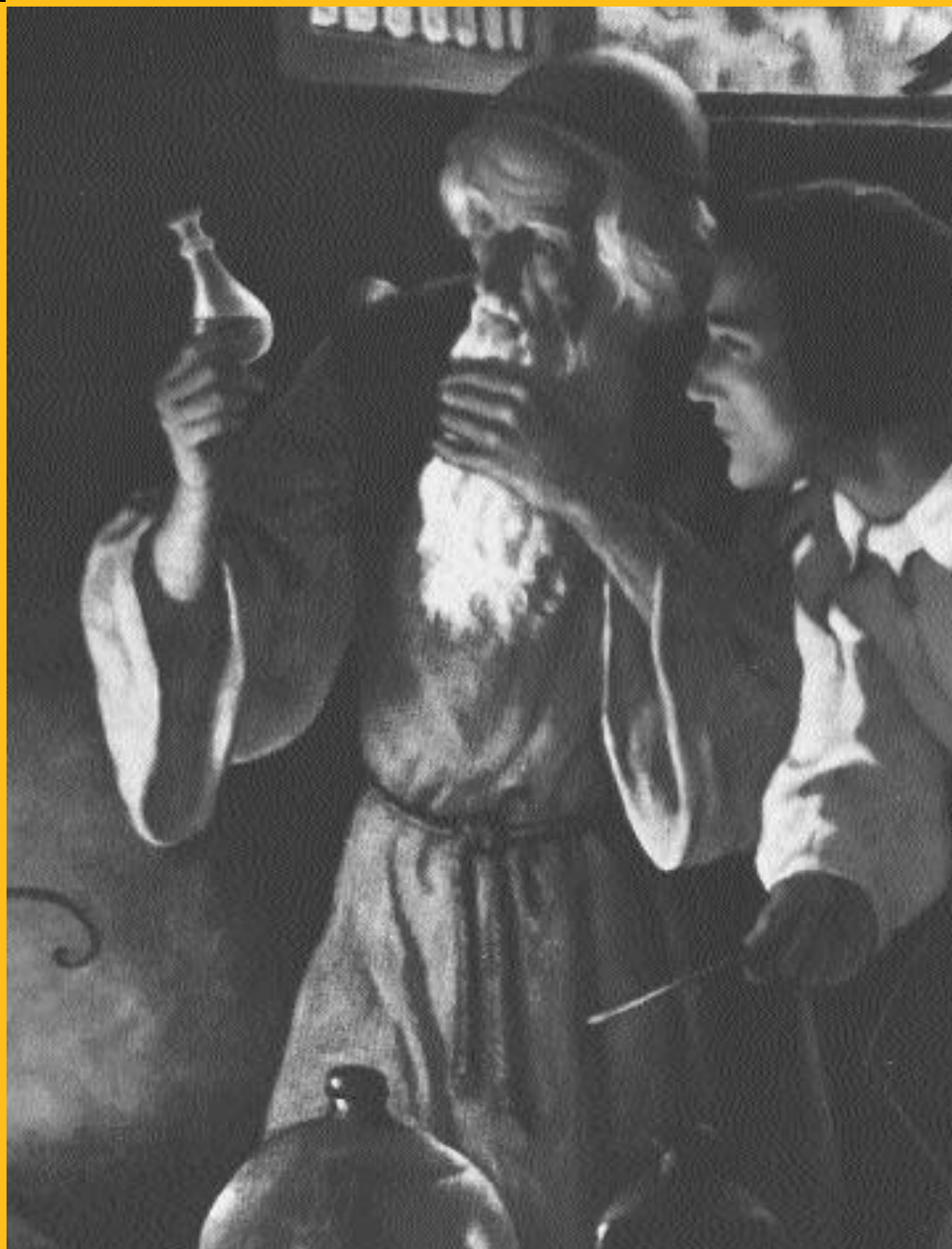
Software Acquisition Best Practices Initiative

THE PROGRAM MANAGER'S GUIDE TO SOFTWARE ACQUISITION BEST PRACTICES



This publication was prepared for the

Software Program Managers Network
4600 North Fairfax Drive, Suite 302
Arlington, VA 22203

The ideas and findings in this publication should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Norm Brown
Director, Software Program Managers Network

This work was created by Computers & Concepts Associates in the performance of Space and Naval Warfare Systems Command (SPAWAR) Contract Number N00039-94-C-0153 for the operation of the Software Program Managers Network (SPMN).

THE PROGRAM MANAGER'S GUIDE TO SOFTWARE ACQUISITION BEST PRACTICES

PREFACE

PREFACE

The Department of Defense and its contractors have arrived at a software crisis. Far too many large-scale software projects have become unaffordable and unable to deliver needed quality, reliability, and capability within the required time frame. Their outputs are not predictable. Their processes are little more than chaotic and do not effectively utilize the kinds of disciplines necessary to achieve success. They have not yet taken advantage of the kinds of practices used to effectively manage large-scale hardware projects.

The Software Acquisition Best Practices Initiative was established to bring about substantial improvements in productivity, quality, timeliness, and user satisfaction by implementing Best Practices as a new foundation for DoD software management. Two of the Initiative's purposes are: focusing the Defense acquisition community on employing effective high-leverage software acquisition management practices; and enabling managers to exercise flexibility in implementing practices within disparate program cultures. Intended for both government software program managers and their industry counterparts, this book is a primary component of the Initiative and has been structured to support both these purposes.

Solutions are taken from successful programs—practices which when effectively implemented, given competent staff, will help bring order, predictability, and higher levels of productivity and quality. Each practice includes key applicability factors, enabling adaptation to particular situations and environments. In addition, the Software Program Managers Network is preparing more detailed information and implementation examples.

These practices are focused upon effective management processes, techniques for finding defects as they occur, eliminating excessive and unnecessary costs, increasing productivity, and other beneficial effects. The Airlie Software Council—nineteen industry leaders, authors, and visionaries—and other industry experts and consultants are convinced that projects effectively utilizing the Principal Best Practices and other appropriate Best Practices will achieve significant cost reductions while simultaneously increasing quality and reliability.

The Airlie Software Council believes the nine Principal Best Practices are applicable to all large-scale projects (i.e., projects relying on the full-time efforts of twelve or more people annually). The Best Practices should be used as appropriate according to the particular circumstances and environment of a given project. All practices are generally applicable to both government and industry projects, and to nearly all domains.

These practices have been through a thorough review and comment process. Both the initial set of Candidate Practices and the later set of Draft Practices were widely distributed for comment. They have been provided to DoD Program Executive Offices, Warfare Centers, industry associations, government and industry participants, and to anyone requesting them as a result of the several best practice presentations at Tri-Ada, the Software Technology Conference, and elsewhere. They have also been made available on the Network's World Wide Web site. As a result, we have received many useful comments. Nearly all recommendations for more clearly articulating a given practice were incorporated. Other recommendations included: conducting additional studies; gathering and including additional information that supports or relates to each practice; providing exact directions on practice use for different program sizes or phase; and suggested major restructuring. These recommendations have all been carefully considered and included in planning for future improvement of this book. The practices herein are a "living set," and are intended to be refined over time. We would greatly appreciate receiving your comments and suggestions (preferably by e-mail). In the event that a program manager desires additional information, the Network will provide source materials and access to experts.

These practices are termed "Best Practices" not because they have been intensively studied and analytically proven to be "best," but simply because they are practices used by, and considered critical to, successful software projects. This is not presumed to be an exhaustive set, or to say that there may not be other, perhaps even better, practices; however, the practices presented here will go a long way to engendering successful software development and maintenance.

This book has been produced by the Initiative under the leadership and funding of the Software Program Managers Network, and was made possible only through the dedicated efforts of some 190 concerned, committed, and experienced software managers, practitioners, leaders, and experts in industry and government that comprised the seven Issue Panels, the Program Managers Panel, and the Airlie Software Council (listed in Appendix D).

The nation's software industry owes a debt of gratitude to Noel Longuemare, Principal Deputy Under Secretary of Defense (A&T), and Emmett Paige Jr., Assistant Secretary of Defense (C3I), for establishing the Initiative and for their commitment to implementing major improvements in software development and maintenance of large-scale software systems. Special thanks are also due to Gene Porter, formerly DoD's Director of Acquisition Program Integration, for focusing the DoD acquisition community on the need for identification and effective use of software acquisition Best Practices.

Norm Brown
Coordinator,
Software Acquisition Best
Practices Initiative

CONTENTS

CONTENTS

THE PROGRAM MANAGER'S GUIDE TO SOFTWARE ACQUISITION BEST PRACTICES

CHAPTER 1

INTRODUCTION TO SOFTWARE ACQUISITION BEST PRACTICES

CHAPTER 1

INTRODUCTION TO SOFTWARE ACQUISITION PROGRAM MANAGEMENT

You know the problems. Headlines proclaim them: big cost overruns, schedule slips, and dramatic performance deficiencies in weapon, C4I, and automated information systems—virtually all due to software problems. In 1987, the Defense Science Board concluded that these software acquisition problems came not from technical difficulties, but from poor management. Since 1987, the situation has gotten worse, not better. Software has gotten bigger, more complex, and more expensive, and ineffective management is still the root cause of much that afflicts software acquisition.

Nowhere is successful software program management more critical than in the Department of Defense. The Assistant Secretary of Defense for Command, Control, Communications and Intelligence estimates annual DoD software maintenance and development expenditures to be $42 billion. Anyone associated with government large-scale software program management knows how rarely a clear program success occurs.

This book provides winning strategies used by successful government and industry software program managers—practices and tools that, if utilized, will enable the effective management of large-scale software programs. In the following chapters, six key components of successful management are examined:

- The PROJECT CONTROL PANEL displays project progress indications and warnings to help gauge how well the project is running.
- The PROJECT ANALYZER consists of diagnostic questions about project status, and helps determine whether further evaluation is called for.
- QUANTITATIVE TARGETS provide hard numbers for production goals, and warnings of possible project malpractice.
- PRINCIPAL BEST PRACTICES are essential to successfully manage software development and maintenance projects in industry and government.
- BEST PRACTICES have been used successfully in industry and government software programs, and are recommended for consideration.
- PROJECT CAVEATS are hard-learned lessons from industry and government software project experience.

In order for the software program manager to maintain control over ever-changing program conditions,

continuing status inputs must be received from every key project element. Just as a pilot would not become airborne without instruments to monitor essential aircraft functions, a program manager should not attempt to manage without effective tools to monitor and control program progress. Effective management of software development and maintenance requires close attention to the latest information on major project issues affecting the bottom line—productivity, quality, timeliness, and user satisfaction.

A conceptual framework of software acquisition management is given in Figure 1.1.

CHAPTER 1

INTRODUCTION TO SOFTWARE ACQUISITION PROGRAM MANAGEMENT

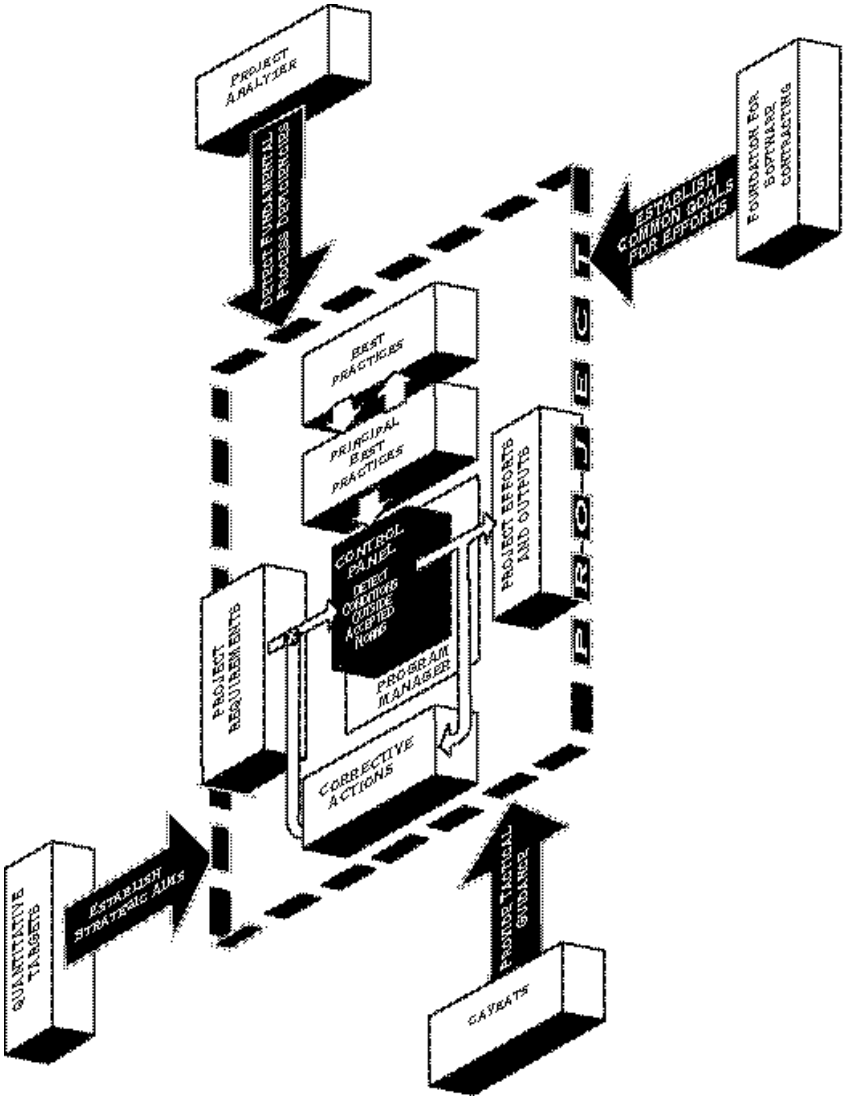


FIGURE 1.1 SOFTWARE ACQUISITION PROGRAM MANAGEMENT ENVIRONMENT

CHAPTER 2

PROJECT CONTROL PANEL

CHAPTER 2

PROJECT CONTROL PANEL

The Project Control Panel is both a concept and a tool for visualizing and monitoring the condition of a project and predicting its future course. The Panel facilitates the entire project team's quick determination of the status of their project, and identification of areas for improvement. The Control Panel was designed to help project managers keep their projects on course when data for the Control Panel is updated regularly. When gauges are not in acceptable ranges, they indicate to management that potential trouble lies ahead. The Control Panel displays information on progress, that including: productivity and completion, change, staff, risk, and quality. These criteria were chosen to cover the primary areas that every project manager needs to track in order to avoid failure on large-scale software development projects.

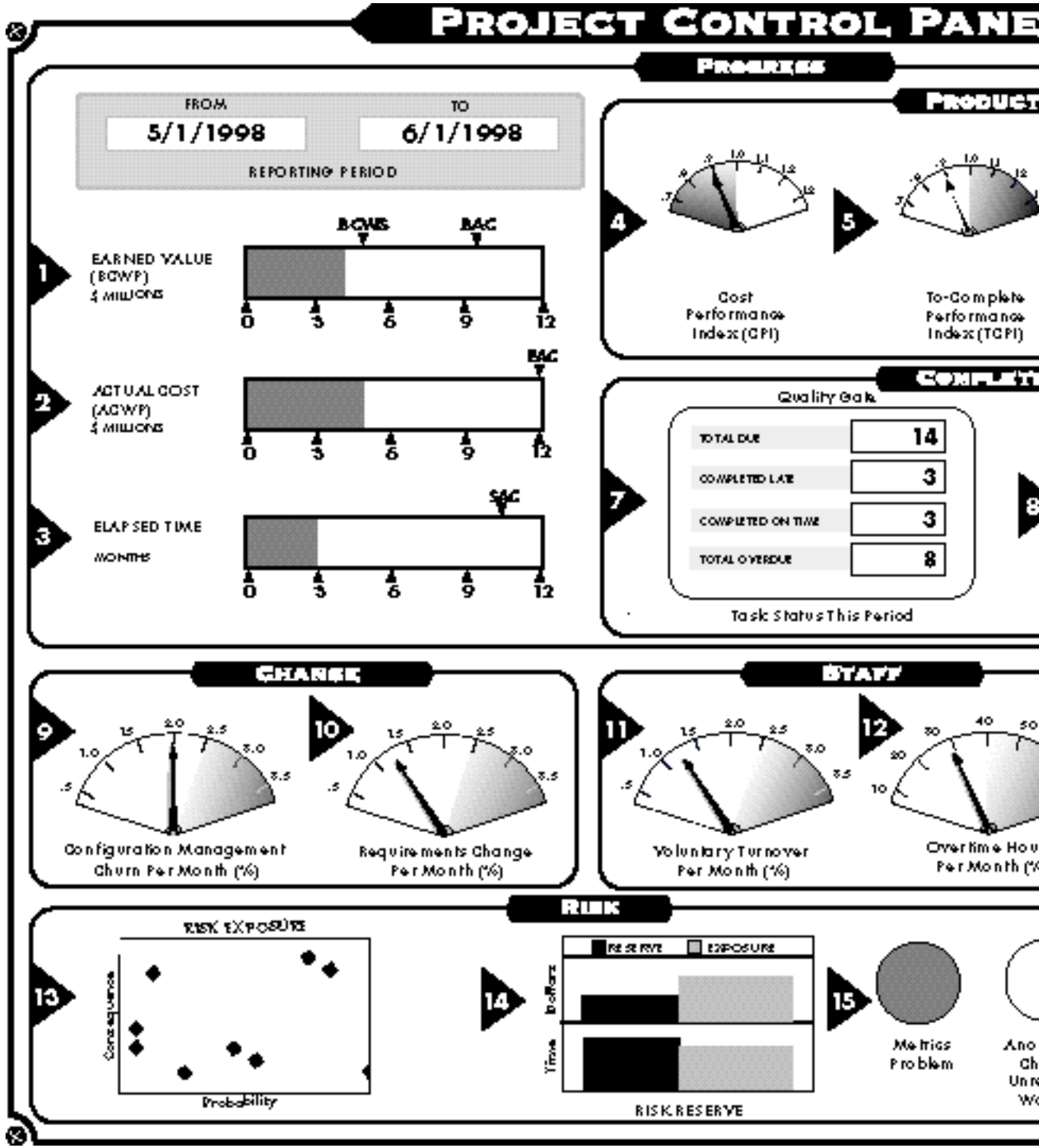
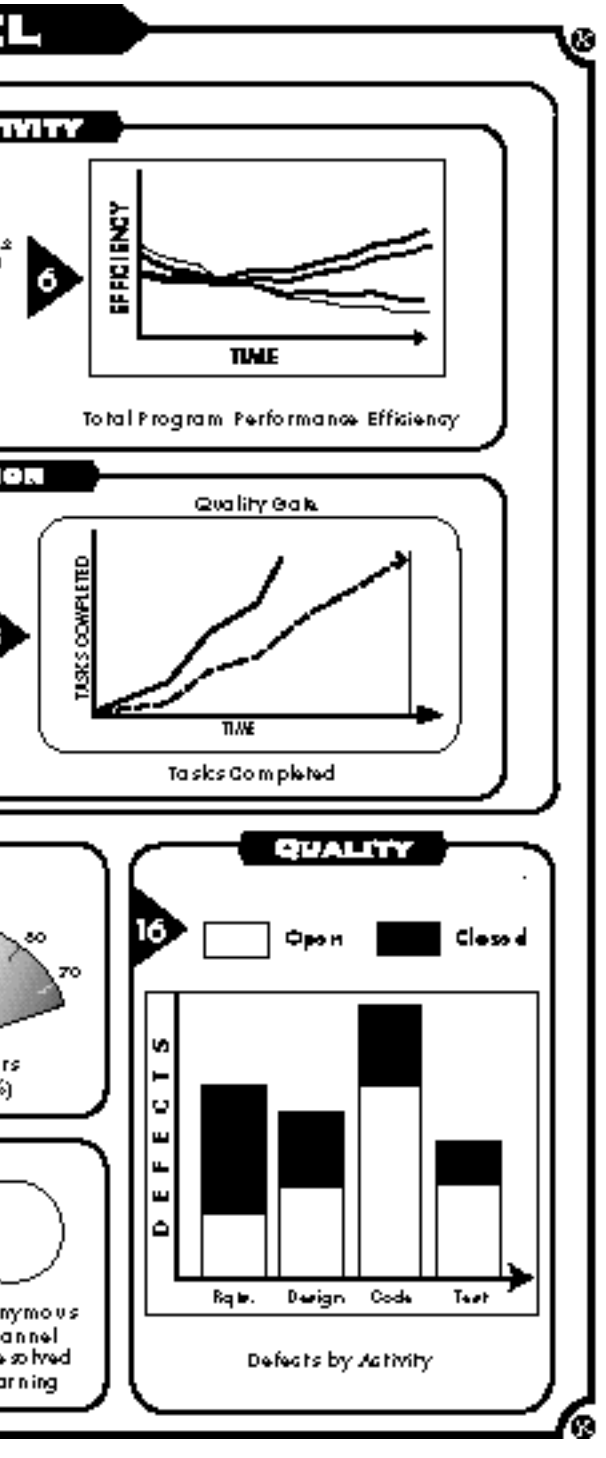


FIGURE 2.1 PROJECT CONTROL PANEL



1. Cumulative Earned Value delivered (BCWP) compared with total budgeted cost (BAC) and cumulative planned value (BCWS).

2. Cumulative actual cost (ACWP) compared with total estimated cost at completion (EAC).

3. Current reporting time period compared with total periods budgeted.

4. $\text{CPI} = \frac{\text{BCWP}}{\text{ACWP}}$

5. $\text{TCPI} = \frac{\text{BAC} - \text{BCWP}}{\text{EAC} - \text{ACWP}}$

6. Total program performance efficiency chart.

7. Number of tasks due, completed on time, completed late, and total overdue tasks last month.

8. Cumulative number of tasks planned and completed over time.

9. $\frac{\text{\# of modified CIs rechecked into CM last month}}{\text{\# of CIs in CM system}} \times 100$

10. $\frac{\text{\# of new and changed requirements last month}}{\text{\# of original requirements}} \times 100$

11. $\frac{\text{\# of staff voluntarily leaving last month}}{\text{\# of staff at beginning of month}} \times 100$

12. $\frac{\text{\# of overtime hours last month}}{\text{\# of base hours}} \times 100$

13. Each risk plotted in regions of high-, moderate-, and low-risk exposure.

14. Risk reserve dollars:
Total cost risk exposures compared with cost risk reserve.

Risk reserve time:
Total schedule risk exposures compared with schedule risk reserve.

15. Metrics Problem – project metrics warning indicator

Anonymous channel warnings – bad news from staff.

16. Total # of severity 1 & 2 defects that are open and closed.

PROGRESS:

1 The **EARNED VALUE** or **Budgeted Cost of Work Performed (BCWP)** gauge shows the cumulative Earned Value delivered to date. The cumulative Earned Value indicator shows the amount of work that has been completed on the project. This metric is based on the notion that, at the beginning of a project, every task is allocated a budget which then becomes its planned value. As work is completed on a task, its budget (or planned value) is "earned" as a quantitative measure of progress. The maximum value on the gauge is the total original budget for the project, which is known as **Budget at Completion (BAC).** Note that BAC is constant for the life of the project, and represents the total value of work to be performed. The triangle indicator shows the cumulative planned value or **Budgeted Cost of Work Scheduled (BCWS),** which is the total value of work that was originally scheduled for completion by the end of this reporting period.

The cumulative Earned Value (BCWP), cumulative planned value (BCWS), and BAC indicators can be compared with one another to make critical observations about progress on the project. By comparing the BCWP indicator with the BCWS indicator, you can determine if the project is ahead of or behind schedule. This is a good measure of schedule deviation because it takes into account the amount of work that was planned to be completed.

Establishing a planned value and a completion criterion for each task before work begins is critical for using the Earned Value metric successfully to measure progress. Cumulative Earned Value is the sum of the planned values for all completed tasks. The best completion criteria for a software task will require that no planned value credit can be taken until all work is completed and tested. These completion criteria are known as ***quality gates.***

2 The **ACTUAL COST** or **Actual Cost of Work Performed (ACWP)** gauge shows the cumulative actual cost incurred on the project to date. **Estimate at Completion (EAC)** is the maximum value on this gauge,

which represents the current best estimate for total cost of the project. Note that EAC might have a different value from BAC in the above Earned Value gauge because better total cost estimates can be made as the project progresses. Therefore EAC may change for different reporting periods.

By comparing cumulative actual cost (ACWP) with the cumulative Earned Value (BCWP) in the above Earned Value gauge, you can estimate how your project is performing against its budget. This shows how well the project is turning actual costs (ACWP) into progress (BCWP). Although the scales for this gauge and the Earned Value gauge are the same, cumulative actual cost can be compared with BAC to determine project status toward over running the original budget, and with EAC to determine project status toward overrunning the current estimated total cost.

3 The **ELAPSED TIME** gauge shows the end date for the current reporting period. The **SAC (Schedule at Completion)** mark shows the original scheduled completion date for the project.

Current time can be compared with SAC to determine the time remaining in the original schedule.

4 The **COST PERFORMANCE INDEX (CPI)** gauge shows how efficiently the project team has turned costs into progress to date. It is calculated by dividing cumulative Earned Value by the cumulative actual cost (BCWP/ACWP). It is a historical measure of average productivity over the life of the project.

CPI represents how much work was performed for each dollar spent, or "bang for the buck." When CPI has a value of 1.0, the project team is delivering a dollar of planned work for each dollar of cost. When CPI is less than 1.0, there is the potential for a productivity problem. For example, a CPI of .80 means that you received 80 cents' worth of planned work for each dollar you paid in cost. A CPI of less than 1.0 may indicate that the project team didn't perform as well as expected, or that the original budget was too aggressive for the amount of work to be performed.

5 The **TO-COMPLETE PERFORMANCE INDEX (TCPI)** gauge shows the future projection of the average productivity needed to complete the project within an estimated budget. It is calculated by dividing the work remaining by the current estimate of remaining cost ((BAC - BCWP)/(EAC - ACWP)).

The TCPI gauge must be used in conjunction with the CPI gauge. TCPI should be compared with CPI to determine how realistic the most recent estimated total cost (EAC) is for the project. Note that CPI measures the average historic productivity to date. If TCPI is greater than CPI, then the project team is anticipating an efficiency improvement to make it more productive. The estimated total cost of the project (EAC) can therefore be calibrated by comparing TCPI with CPI. Always question claims of future productivity improvement that result in a 20 percent or greater increase in TCPI over CPI in order to ensure they are based on sound reasoning. This is especially true of "silver bullets" like new tools, languages, or methodologies that may actually decrease

productivity due to training and start-up costs. The redline on this gauge should be about 20 percent above the current value of the CPI gauge to show the relationship and warning level between the two gauges.

6 The **ABBA[1] CHART**, also known as a **Total Program Performance Efficiency** chart, is composed of four different performance indicators showing trends in historic and projected efficiency to date. The four indicators are:

- TCPI (Gauge 5)
- Completion Efficiency (CE), a ratio calculated by dividing BAC by EAC to estimate the productivity required to complete the project within a projected total cost (EAC)
- CPI (Gauge 4)
- Monthly CPI, a ratio calculated by dividing the monthly Earned Value by the monthly actual cost (as opposed to cumulative values for the CPI calculation)

A more detailed description of this chart is presented in Appendix C.

7 **QUALITY GATE TASK STATUS THIS MONTH** shows the completion status of tasks during the current reporting period. A quality gate is a predefined completion criterion for a task. The criterion must be an objective yes/no indicator that shows a task has been completed (see discussion on Gauge 1 above). The indicators are:

- **Total Due** is the total number of tasks scheduled for completion during the current reporting period plus any overdue tasks from previous periods. This indicates the total quantity of work required for the project to keep pace with the schedule.
- **Completed On Time** is the number of tasks originally scheduled for completion during the current reporting period that were completed by their original scheduled due date. This number indicates how well the project is keeping up with scheduled work.
- **Completed Late** is the number of tasks completed late during the current reporting period. This number includes those tasks scheduled for the current period that were completed late, as well as any overdue tasks from previous periods that were completed in the current period. The Completed Late number indicates how well the project is completing work, even if it is late according to the original schedule.

[1] Named for Wayne Abba of the Department of Defense.

- **Total Overdue** is the total number of tasks for all previous reporting periods that are overdue by the end of the current reporting period. This is an indicator of the quantity of work needed to get the project back on schedule.

The total number of tasks completed in the current reporting period is the sum of Completed On Time and Completed Late. Total Overdue is equal to Total Due minus Completed On Time and Completed Late.

8 The **QUALITY GATE TASKS COMPLETED** graph shows the cumulative number of tasks completed by the end of each reporting period to date plotted with the cumulative number of tasks scheduled for completion.

When the number of tasks completed is less than the number planned, then the horizontal difference on the time axis is an indicator of the current schedule slip to date.

CHANGE:

9 **CM (CONFIGURATION MANAGEMENT) CHURN PER MONTH** is calculated by taking the number of baselined Configuration Items (CIs) that have been modified and rechecked into the Configuration Management system over the last reporting period and dividing it by the total number of baselined CIs in the system at the end of the period. It is expressed as a percentage. A modified CI is one that was previously in the system, but was reviewed sometime later and then modified or replaced.

This gauge serves as an indicator of the architectural soundness of the system. If the rate of "churn" begins to approach the 2-percent-per-month level, this shows that a lot of rework is going on, which could point to deeper problems in the project. A high churn rate may mean that the original design was not robust enough. It could also be a symptom of changing requirements (see Gauge 10), which could indicate the project is drifting towards disaster

10 **REQUIREMENTS CHANGE PER MONTH** is calculated by dividing the number of new, changed, or deleted requirements specified in the current reporting period by the total number of requirements at the end of the current period. It is expressed as a percentage. Typical projects experience a requirements change of 1 percent per month.

Some requirements growth is to be expected, particularly on large projects. However, a high rate of requirements change can indicate the customer is not sure of what is wanted, or the original requirements definition was poor. A high rate often predicts disaster for software-intensive projects.

STAFF:

11 **VOLUNTARY TURNOVER PER MONTH** is calculated by dividing the number of staff leaving during the current reporting period by the number of staff at the beginning of the current period. It is expressed as a percentage. The target range is less than 2

percent per month. A person can leave the project in a number of ways, such as by quitting the organization or requesting reassignment to another project.

Turnover is an important measure for risk assessment. Every project lasting six months or longer should expect and prepare for some staff turnover. Each project member who leaves the team causes a productivity drop and schedule disruption. However, bringing on new team members, regardless of skills and experience, does not necessarily solve the problem; they require time to become familiar with the project and processes. In addition, a productive team member will usually have to devote time to orient the new hire, thus taking away additional resources from the project. Appropriate allowances should be included in the productivity resource estimates to allow for staff turnover

12 **OVERTIME PER MONTH** is calculated by dividing the overtime hours by the base working hours for all project staff in the current reporting period. It is expressed as a percentage. The target range is less than 10 percent. When the overtime rate approaches 20 percent, the ability of the staff to respond effectively to crises suffers significantly.

RISK:

13 The **RISK EXPOSURE** chart shows each risk plotted by its cost consequence and probability. The probability is expressed in terms of occurrences over the life of the project. The regions on the graph show where risks fall into areas of low-, moderate-, or high-risk exposure.

14 **RISK RESERVE** shows the total risk exposure for cost and schedule compared with the current cost and time risk reserves for the project. Risk exposure for a risk is calculated by multiplying the probability by the consequence of that risk. Although the consequences (and therefore the risk exposure) for all risks are not necessarily independent, a first approximation to the total cost risk exposure to the project can be made by summing the individual cost risk exposures for all risks. This same rule holds true for consequences resulting in a delayed schedule.

A cost and risk reserve should be established at the beginning of the project to deal with unforeseen

problems. The cost and time risk reserve for a project will change over time as some of this reserve is used to mitigate the effects of risks that actually occur and affect the project.

15 The **METRICS PROBLEM** indicator shows that project management has received either a warning or bad news about some of the metrics on the project.

The **ANONYMOUS CHANNEL UNRESOLVED WARNING** indicator shows that project management has received either a warning or bad news about the actual status of the project.

An open-project culture in which reporting bad news is encouraged is conducive to a healthy project. Warnings from anonymous or known project personnel should be welcomed and tracked.

QUALITY:

16 **DEFECTS BY ACTIVITY** displays the number of detected defects open (i.e., yet to be fixed) and the number of defects closed in each phase of the project. Defects are problems that, if not removed, could cause a program to fail or produce incorrect results. Defects are generally prioritized by severity level, with those labeled as numeral 1 being the most serious.

The quality indicators on this chart help you answer the question, "What is the quality of the product right now?"

CHAPTER 3

PROJECT ANALYZER

CHAPTER 3

PROJECT ANALYZER

The Project Analyzer questions provide program managers with a "quick look" at software project health. The Project Analyzer determines whether key program elements exist, without which the program is not likely to succeed. If a program manager cannot answer the following questions about current project status, or must answer in the negative, the project should be scheduled for immediate review.

1. **Do you have a current, credible activity network supported by a Work Breakdown Structure (WBS)?**
2. **Do you have a current, credible schedule and budget?**
3. **Do you know what software you are responsible for delivering?**
4. **Can you list the current top ten project risks?**
5. **Do you know your schedule compression percentage?**
6. **What is the estimated size of your software deliverable? How was it derived?**
7. **Do you know the percentage of external interfaces that are not under your control?**
8. **Does your staff have sufficient expertise in the key project domains?**
9. **Have you identified adequate staff to allocate to the scheduled tasks at the scheduled time?**

1. Do you have a current, credible activity network supported by a Work Breakdown Structure (WBS)?

An activity network is the primary means to organize and allocate work.

- Have you identified your critical path items?
- What explicit provisions have you made for work that isn't on your WBS?
- Does the activity network clearly organize, define, and graphically display the work to be accomplished?
- Does the top-level activity network graphically define the program from start to finish, including dependencies?
- Does the lowest-level WBS show work packages with measurable tasks of short duration?
- Are project objectives fully supported by lower-level objectives?
- Does each task on the network have a well-defined deliverable?
- Is each work package under budget control (expressed in labor hours, dollars, or other numerical units)?

A well-constructed activity network is essential for accurate estimates of project time, cost, and personnel needs, because estimates should begin with specific work packages.

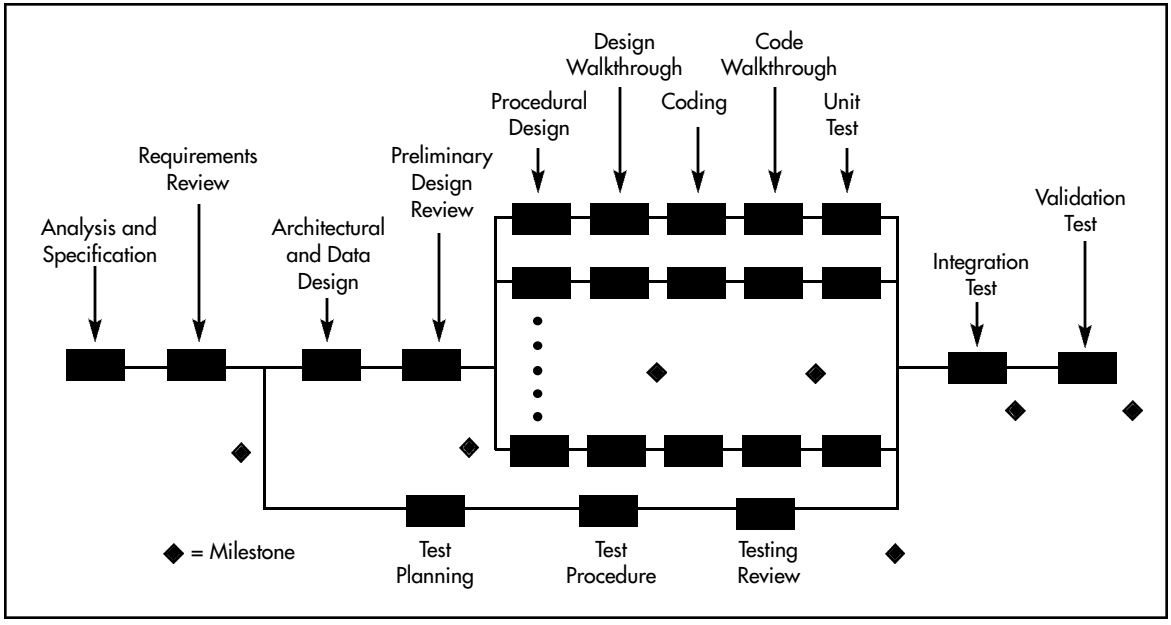


FIGURE 3.1 ACTIVITY NETWORK EXAMPLE

2. Do you have a current, credible schedule and budget?

- Is the schedule based on a project/activity network supported by the WBS?
- Is the schedule based on realistic historical and quantitative performance estimates?
- Does the schedule provide time for education, holidays, vacations, sick leave, etc.?
- Does the schedule provide time for quality assurance activities?
- Does the schedule allow for all interdependencies?
- Does the schedule account for resource overlap?
- Is the schedule for the next 3-6 months as detailed as possible?
- Is the schedule consistently updated at all levels on Gantt, PERT, and Critical Path charts every two weeks?
- Is the budget clearly based on the schedule and required resources over time?
- Can you perform to the schedule and budget?

3. Do you know what software you are responsible for delivering?

- Are system operational requirements clearly specified?
- Are definitions of what the software must do to support system operational requirements clearly specified?
- Are system interfaces clearly specified and, if appropriate, prototyped?
- Is the selection of software architecture and design method traceable to system operational characteristics?
- Are descriptions of the system environment and relationships of the software application to the system architecture specified clearly?
- Are specific development requirements expertly defined?
- Are specific acceptance and delivery requirements expertly defined?
- Are user requirements agreed to by joint teams of developers and users?
- Are system requirements traceable through the software design?

4. Can you list the current top ten project risks?

- Has a full-time Risk Management Officer been assigned to the project?

- Are risks determined through established processes for risk identification, assessment, and mitigation?
- Is there a database that includes all non-negligible risks in terms of probability, earliest expected visible symptom, and estimated and actual schedule and cost effects?
- Are all project personnel encouraged to become risk identifiers? Is there an anonymous communications channel for transmitting and receiving bad news?
- Are correction plans written, followed up, and reported?
- Is the database of top ten risks updated regularly?
- Are transfers of all deliverables/products controlled?
- Are user requirements reasonably stable?
- How are risks changing over time?

5. Do you know your schedule compression percentage?

- Has the schedule been constructed bottom up from quantitative estimates, not by predetermined end dates?
- Has the schedule been modified when major software modifications have taken place?
- Have programmers and test personnel received training in the principal domain area, the hardware, support software, and tools?
- Have very detailed unit-level and interface design specifications been created for maximum parallel programmer effort?
- Does the project avoid extreme dependence on specific individuals?
- Are people working abnormal hours?
- Do you know the historical schedule compression percentage on similar projects, and the results of those projects?
- Is any part of the schedule compression based on the use of new technologies?
- Has the percent of software functionality been decreased in proportion to the percent of schedule compression?

$$\text{Schedule Compression Percentage} = \left\{ 1.00 - \left[\frac{\text{Calendar Time Scheduled}}{\text{Nominal Expected Time}} \right] \right\} \times 100$$

Nominal Expected Time is a function of total effort expressed in person months.

For example, Boehm[1] found that:

For a class of DoD project of 500 person months or more:

$$\text{Nominal Expected Time} = 2.15 \times \left[\text{Expected Person Months} \right]^{0.33}$$

Nominal Expected Time was measured from System Requirements Review to System Acceptance Test.

[1] *Software Engineering Economics*, Prentice Hall, 1981.

Attempts to compress a schedule to less than 80 percent of its nominal schedule aren't usually successful.

New technologies are an additional risk in time and cost.

6. What is the estimated size of your software deliverable? How was it derived?

- Has the project scope been clearly established?
- Were measurements from previous projects used as a basis for size estimates?
- Were Source Lines of Code (SLOC) used as a basis for estimates?
- Were Function Points (FPs) used as a basis for estimates?
- What estimating tools were used?
- Are the developers who do the estimating experienced in the domain area?
- Were estimates of project size corroborated by estimate verification?
- Are estimates regularly updated to reflect software development realities?

Software size estimation is a process that should continue as the project proceeds.

7. Do you know the percentage of external interfaces that are not under your control?

- Has each external interface been identified?
- Have critical dependencies of each external interface been documented?
- Has each external interface been ranked based on potential project impact?
- Have procedures been established to monitor external interfaces until the risk is eliminated or substantially reduced?
- Have agreements with the external interface controlling organizations been reached and documented?

8. Does your staff have sufficient expertise in the key project domains?

- Do you know what the user needs, wants, and expects?
- Does the staffing plan include a list of the key expertise areas and estimated number of personnel needed?

- Does most of the project staff have experience with the specific type of system (business, personnel, weapon, etc.) being developed?
- Does most of the project staff have extensive experience in the software language to be used?
- Are the developers able to proceed without undue requests for additional time and cost to help resolve technical problems?
- Do the developers understand their project role, and are they committed to its success?
- Are the developers knowledgeable in domain engineering—the process of choosing the best model for the project and using it throughout design, code, and test?
- Is there a domain area expert assigned to each domain?

9. Have you identified adequate staff to allocate to the scheduled tasks at the scheduled time?

- Do you have sufficient staff to support the tasks identified in the activity network?
- Is the staffing plan based on historical data of level of effort or staff months on similar projects?
- Do you have adequate staffing for the current tasks and all the tasks scheduled to occur in the next two months?
- Have alternative staff buildup approaches been planned?
- Does the staff buildup rate match the rate at which the project leaders identify unsolved problems?
- Is there sufficient range and coverage of skills on the project?
- Is there adequate time allocated for staff vacations, sick leave, training, and education?
- Are staffing plans regularly updated to reflect reality?

CHAPTER 4

QUANTITATIVE TARGETS

CHAPTER 4

QUANTITATIVE TARGETS

Quantitative targets apply to key project areas being measured, providing best-in-class objectives for DoD-contracted software projects. The targets and their associated warning levels of possible malpractice are:

- Original Defect Density and Defect Removal Efficiency
- Slip or Cost Overrun in Excess of Risk Reserve
- Total Requirements Growth
- Total Software Program Documentation
- Voluntary Staff Turnover Per Year

SIZE METRIC ALTERNATIVES

Both Source Lines of Code (SLOC) and Function Points (FPs) can be used as basic measurements of software size and complexity. The following enables SLOC to be converted to FPs, and vice versa:

PROGRAMMING LANGUAGE	SLOC PER FUNCTION POINT		
	LOW	MEAN	HIGH
Assembly language	200	320	450
C	60	128	170
C++	20	29	65
Jovial	75	106	160
CMS-2Y	75	106	160
ALGOL, CHILL, COBOL, FORTRAN	75	106	160
Pascal	50	91	125
RPG, PL/I	50	80	115
Modula 2 Ada	70	80	90
Prolog, LISP, Forth, BASIC	35	64	90
Logo	----	53	----
Fourth generation database	----	40	----
Stratagem	----	35	----
APL	----	32	----
Objective-C	17	26	38
SmallTalk	12	21	30
Query languages	----	16	----
Spreadsheet languages	3	6	9

FIGURE 4.1 SLOC-TO-FUNCTION-POINT CONVERSION TABLE
Reprinted from *Applied Software Measurement*, Capers Jones, McGraw-Hill, 1991.

Measurement	Target	Warning Level
Defect Removal Efficiency	>95%	<85%
Original Defect Density	<4 Per Function Point	>7 Per Function Point

Metrics can provide valuable insight into a software development process and are also valuable for process improvement.

A **DEFECT** is a "bug" or problem which, if not removed, could cause a computer program to either fail or produce incorrect results.

DEFECT REMOVAL EFFICIENCY is the aggregate of all defects removed by all means, to include:

- Desk checking
- Reviews
- Walkthroughs
- Inspections
- Editing

In order to calculate defect removal efficiency, defect totals are tracked from requirements specification through system delivery and into the first year of being fielded.

$$\text{Removal Efficiency} = \frac{\text{Prerelease Defects}}{\text{Prerelease defects + one year of reported defects}}$$

Defect Origins	Defect Potential	Removal Efficiency	Delivered
Requirements	1.00	77%	0.23
Design	1.25	85%	0.19
Coding	1.75	95%	0.09
Document	0.60	80%	0.12
Bad Fixes	0.40	70%	0.12
TOTAL	**5.00**	**85%**	**0.75**

Figure 4.2 Defect Removal Efficiency for Selected Defect Removal Activities
Reprinted from *Applied Software Measurement,* Capers Jones, McGraw-Hill, 1991.

Techniques and technologies used to minimize or prevent the risk of human error by software engineering staff include:

- Formal quality plans
- Use of Joint Application Design (JAD) techniques, including intensive team-based analysis, design, and development sessions
- Prototyping

- Reusable designs and code from certified sources
- Software quality assurance teams
- Total quality management methods
- Reviews, walkthroughs, and inspections
- Use of appropriate automated defect estimation and measurement tools
- Clean Room development methods —a model using statistical quality control to engineer software (See Best Practice "Encourage Clean Room Techniques," Chapter 6, for applicability.)

A dollar spent on defect prevention has been shown to reduce the costs associated with fixing defects from $70.00 to $3.00 per Function Point.

MEASUREMENT	TARGET	WARNING LEVEL
Schedule Slip or Cost in Excess of Risk Reserve	0%	≥ 10%

SLIP is the amount of time that a deliverable or product is late from its originally scheduled date.

COST OVERRUN refers to projects or deliverables in which the actual cost exceeds the estimated or budgeted amounts.

The **SCHEDULE** outlines the time and lists the milestones from requirements specification to product acceptance.

RISK RESERVE is money and time held in reserve to be used in the event that risks occur.

Preventive techniques include using:

- Functional metrics to quantify estimates and progress toward plans
- Commercial-grade software project estimating tools
- Commercial-grade project planning tools
- A measurement process that includes lowest-level activity, or inch-pebble, measurement

Software development schedule slips and cost overruns tend to increase in exponential proportion to project size.

Cost overruns occur on more than 50 percent of all software projects that have more than 25,000 Function Points.

Measurement	Target	Warning Level
Total Requirements Growth (In FP or Equivalent)	≤ 1% Per Month	≥ 50% Per Year

REQUIREMENTS GROWTH is defined as the increase between baselined and current documented requirements.

At 1 percent growth per month, system requirements on a 3-year project will increase baseline requirements by 33 percent.

Techniques for establishing a more effective requirements process include:

- Joint Application Design (JAD) which includes intensive team-based analysis, design, and development sessions
- Integrated Product Teams (IPTs) composed of clients, technical staff, key support groups, and contractor staff
- Prototyping
- Formal change control processes

Unacceptable requirements growth occurs on more than 70 percent of projects over 1,000 Function Points.

Measurement	Target	Warning Level
Total Software Program Documentation	<1000 Words Per FP	>2000 Words Per FP

PROGRAM DOCUMENTATION includes all online and hardcopy information supporting the system's contractual agreement, design, build, operation, and maintenance.

Paperwork can be reduced by:

- Decreasing documentation required by standards
- Decreasing delivery of documentation that already exists in development files
- Using outlines and guidelines for specification and design documentation
- Observing standards for brevity and clarity in paperwork requirements
- Building and using a database specifically for documentation and publication

Fifty-two percent of the total costs of defense systems is attributable to paperwork.

MEASUREMENT	TARGET	WARNING LEVEL
Voluntary Staff Turnover Per Year	1-3%	≥ 10%

Human continuity is important to project success.

VOLUNTARY STAFF TURNOVER is a measurement of employees the project wants to keep, but who choose to leave. A small amount of staff turnover is expected on projects that last longer than six months.

Common reasons why software engineering employees choose to leave are:

- Tough economic conditions resulting from smaller defense budgets
- Having undervalued skills and a market for their skill
- Poor project management
- Inadequate tools
- Unsatisfactory work conditions, location, environment, unreasonable hours, etc.

PLANNED STAFF DURATION (WEEKS)	TERMINATION EFFECT (WEEKS)	REPLACEMENT DELAY (WEEKS)	ASSIMILATION EFFECT (WEEKS)	TOTAL (WEEKS)	LOSS % (WEEKS X 2)
20	2	4	4	10	50
40	2	4	4	10	25
80	4	4	6	14	17
120	4	4	6	14	11

FIGURE 4.3 IMPACT OF STAFF TURNOVER ON PLANNED PROJECT PRODUCTIVITY
Reprinted from *Software Costing*, Frank Wellman, Prentice Hall, 1992.

CHAPTER 5

PRINCIPAL BEST PRACTICES

CHAPTER 5

PRINCIPAL BEST PRACTICES

Under the aegis of the Software Program Managers Network, the Airlie Software Council of software experts and industry leaders has identified nine Principal Best Practices that, if implemented, will improve software development and maintenance productivity and quality, reduce cost, and improve user satisfaction. These practices, which have proven successful in industry, are applicable to nearly all large-scale DoD software development projects:

1. Formal Risk Management
2. Agreement on Interfaces
3. Formal Inspections
4. Metrics-based Scheduling and Management
5. Binary Quality Gates at the Inch-Pebble Level
6. Program-wide Visibility of Progress vs. Plan
7. Defect Tracking Against Quality Gates
8. Configuration Management
9. People-Aware Management Accountability

1. FORMAL RISK MANAGEMENT

A formal risk management process requires corporate acceptance of risk as a major consideration for software program management, commitment of program resources, and formal methods for identifying, monitoring, and managing risk.

Problem Addressed:

All software has risk. The cost of resolving a risk is usually relatively low early on, but increases dramatically as the project progresses.

Practice Essentials:

- Identify risk
- "Decriminalize" risk
- Plan for risk
- Formally designate a Risk Officer (a senior member of the management team responsible for risk management)
- Include in the budget and schedule a calculated risk reserve buffer of time, money, and other key resources to deal with risks that materialize
- Compile a database for all nonnegligible risks
- Include technical, supportability, programmatic, cost, and schedule risks
- Prepare a profile for each risk (consisting of probability and consequence of risk actualization)
- Include risks over full life cycle (not just during your watch)
- Do not expect to avoid risk actualization
- Keep risk resolution and workarounds off the critical path by identifying and resolving risk items as early as possible
- Provide frequent Risk Status Reports to program manager that include:
  - Top ten risk items
  - Number of risk items resolved to date
  - Number of new risk items since last report
  - Number of risk items unresolved
  - Unresolved risk items on the critical path
  - Probable cost for unresolved risk vs. risk reserve

Status Checks:

- Has a Risk Officer been appointed?
- Has a risk database been set up?
- Do risk assessments have a clear impact on program plans and decisions?
- Is the frequency and timeliness of risk assessment updates consistent with decision updates during the project?
- Are objective criteria used to identify, evaluate, and manage risks?
- Do information flow patterns and reward

criteria within the organization support the identification of risk by all project personnel?

- Are risks identified throughout the entire life cycle, not just during the current program manager's assignment?
- Is there a management reserve for risk resolution? (See Best Practice "Establish Management Reserves for Risk Resolution," Chapter 6.)
- Is there a risk profile drawn up for each risk, and is the risk's probability of occurrence, consequences, severity, and delay regularly updated?
- Does the risk management plan have explicit provisions to alert decision makers upon a risk becoming imminent?

2. AGREEMENT ON INTERFACES

A baseline interface must be agreed upon before the beginning of implementation activities, and the user interface must be made and maintained as an integral part of the system specification. For those projects developing both hardware and software, a separate software specification must be written with an explicit and complete interface description.

PROBLEM ADDRESSED:

System interfaces generally constitute essential elements of a system's requirements and architecture, but are not completely controlled by the developer. Not ensuring that external interfaces are properly identified, integrated, and stabilized early will create the need for expensive and time-consuming "fixes" later.

PRACTICE ESSENTIALS:

- Recognize that both user interfaces and external system interfaces are critical
- Fully identify and baseline the user interface before beginning development
- Define each input/output data item
- Display navigation between screens as well as screen fields
- Include the user interface as part of system specification
- Use rapid prototyping of a Graphical User Interface (GUI) as a tool for the user to define requirements
- For embedded systems, prepare a separate system specification for the software

STATUS CHECKS:

- Is there a complete census of input/outputs? Are such inputs/outputs defined down to the data element level?
- Are the interfaces stable?
- Have you considered hardware/software, users, major software component interfaces, etc.?
- Have existing and future interfaces been defined, including consideration of those that may be required over time?
- Does the system specification include a separate software specification to show the hardware interfaces?
- Are opportunities made available for users to provide input and review the user interface descriptions as they develop?

Only the prospective operational user can define/verify user interface correctness with a high probability of success (the developer cannot).

For most software, the user interface defines user requirements from both the user and development perspectives.

3. STRUCTURED PEER REVIEWS

Peer reviews should be conducted on requirements, architecture, designs at all levels, code prior to unit test, and test plans.

PROBLEM ADDRESSED:

Rework to fix defects accounts for between 40 percent and 50 percent of total development costs. Structured peer reviews typically find 80 percent of defects as they happen (walkthroughs typically find 60 percent). When effectively used, structured peer reviews can make an enormous difference to program cost, schedule, and quality.

PRACTICE ESSENTIALS:

- Use structured peer reviews starting early in development to effectively identify requirements defects
- Have the customer participate in peer reviews
- Use small teams of prepared reviewers with assigned roles
- Ensure that entry and exit criteria exist for each review

STATUS CHECKS:

- Are peer reviews identified and implemented to assess the quality of all baselined artifacts and placed under control before they are released for project use?
- Is the conduct of peer reviews structured, and are they integrated into the project schedule?
- Are procedures, standards, and rules for the conduct of peer reviews established?

- Are metrics used to gauge the effectiveness of peer reviews?
- Is there a documented process for conducting peer reviews?
- Are entrance and exit criteria established for each peer review?
- Are a significant number of defects caught as early as possible (prior to testing at least)?
- Are peer reviews specifically focused on a narrow set of objectives, and do they evaluate a fixed set of data?
- Is there a clear rationale for the scheduling of peer reviews?
- Are defects from peer reviews tracked and catalogued?
- Are peer reviews conducted to assess the quality of all engineering data products before they are released for project use?
- Is the detailed design reviewable?

4. METRICS-BASED SCHEDULING AND MANAGEMENT

Cost and schedule estimates should be based on empirical data. Metrics-based planning requires early calculation of size, projection of costs and schedules from empirical patterns, and tracking of project status through the use of captured-result metrics.

PROBLEM ADDRESSED:

The important issue here is to *identify problems early*This is the primary reason to make sure metrics are being done right. Your metrics are the yardstick for measuring progress against your baseline plan, and become your warning indicator for further inquiry and action. Of course, the earlier the visibility of a problem, the better the chance of avoiding the problem or controlling its negative effect.

PRACTICE ESSENTIALS:

- Estimate cost and schedule using data from completed projects of similar size and objective
- Compare with cost model estimate
- Plan short-duration tasks with measurable products (see Best Practice "Activity Planning," Chapter 6, and Principal Best Practice "Binary Quality Gates at the Inch-Pebble Level," Chapter 5.)
- Review the following at frequent intervals throughout the project:
  - Earned Value (BCWP) vs. Actual Expended
  - Cost to Complete (including estimate for unresolved risk) vs. Planned at Completion
  - Schedule to Complete vs. Planned Schedule

- Cost Performance Index
- To-Complete Performance Index

- Manage defect closure time (see Principal Best Practice "Defect Tracking Against Quality Targets," Chapter 5.)
- Don't hide problems with rebaselining
- Report Earned Value and other progress measures against original baseline
- Track other Control Panel metrics (see Chapter 2.)

STATUS CHECKS:

- Are cost and schedule performance tracked against the initial baseline and the latest baseline?
- Are the number of changes to the initial cost/schedule baseline tracked?
- Does the plan identify progress measures to permit rate charting and tracking?
- Are inspection coverage and error removal rates tracked for the entire product and for each component?
- Are project estimates continuously refined as the project proceeds?
- Is a project feedback loop established between project measures and updated schedules?
- Is there a process for capturing the primitive data necessary to calculate Earned Value?
- Are productivity levels and schedule deadlines evaluated against past performance and reflected in the risk assessment?
- Are the planned vs. actual cost and planned vs. actual schedule monitored?
- Is there automated support for metrics-based scheduling and tracking procedures?

It's important to determine whether an indicated schedule delay is a problem with the people carrying out the efforts or with a plan that was too aggressive to begin with. That is, don't punish competent, highly productive development staff if they don't meet unreasonable cost and schedule estimates.

5. BINARY QUALITY GATES AT THE INCH-PEBBLE LEVEL[1]

Completion of each task in the lowest-level activity network needs to be defined by an objective binary indication. These completion events should be in the form of gates that assess either the quality of the products produced or the adequacy and completeness of the finished process.

PROBLEM ADDRESSED:

When planning and project monitoring are not based on sufficient detail, any picture of where the program is and how it is progressing is simply an illusion.

By focusing on detail, specific development or maintenance efforts can be more effectively identified, planned, and tracked. By utilizing quality gates that prevent effort outputs from moving on until they pass all their predefined acceptance criteria, and binary determination of the effort's completeness (it's either done or it's not), evaluation of how actual progress is being

[1] The Aerospace Industries Association (AIA) has commented that the use of technical reviews, tests, demonstrations, or audits as "completion criteria for 'inch-pebbles' is excessive in terms of value added to the customer or the contractor." AIA recommended the "retention of the 'Binary Quality Gates' concept and deletion of references to the granularity of the tasks and to the term 'inch-pebbles.'"

made against the plan becomes meaningful.

PRACTICE ESSENTIALS:

- Ensure visibility of where the development really is, based upon products produced
- Ensure that every lowest-level (i.e., inch-pebble) task:
  - Is of short duration
  - Expends a small percent of the total budget
  - Is dedicated to producing a tangible product necessary for a required deliverable
- Define a binary gate for every inch-pebble task (objective acceptance criteria/tests for determining whether the output product is acceptable)
- Give no Earned Value credit for an inch-pebble task until the binary gate is passed

STATUS CHECKS:

- Have credible project status and planning estimates been produced based on inch-pebble quality gates that can be aggregated at any desirable level?
- Have all activities been decomposed into inch-pebbles?
- Has all near-term work been decomposed into tasks no longer than two weeks in duration?
- Have achievable accomplishment criteria been identified for each task? Are tasks based on overall quality goals and criteria for the project?
- Are quality gates rigorously applied for determining task accomplishment, without exception?
- Is there clear evidence that planned tasks are 100 percent complete before acceptance?
- Is there clear evidence of successful completion of inspections?
- Are inch-pebble tasks on the critical path defined, enabling more accurate assessment of schedule risks and contingency plans?
- Is the set of binary quality gates compatible with the WBS?

6. PROGRAM-WIDE VISIBILITY OF PROGRESS VS. PLAN

The core indicators of project health or dysfunction—the Control Panel indicators—should be made readily available to all project participants. Anonymous channel feedback should

be encouraged to enable bad news to move up and down the project hierarchy.

Problem Addressed:

When everyone is involved in identifying problems early, the likelihood of missing problems is greatly reduced, improving risk management and increasing the probability of program success.

Practice Essentials:

- Make Control Panel metrics continuously available to all members of the team and the customer
- Establish an anonymous communications channel for anyone to report problems (This channel will also be used by malcontents, but it's much better to get false alarms than miss a major problem until it is too late to recover.)
- Maintain top-down, program-wide visibility to reduce the number of reports of non-problems

Status Checks:

- Are status indicators on the Control Panel updated at least monthly?
- Are the status indicators integrated into the management decision process?
- Is project status known by all project personnel?
- Can staff report problems as well as successes?
- Are project goals, plans, schedules, and risks available to the project team and interested parties?
- Is anonymous channel feedback visible to all project members?

7. DEFECT TRACKING AGAINST QUALITY TARGETS

Defects should be tracked formally at each project phase or activity. Configuration Management (CM) enables each defect to be recorded and traced through to removal. In this approach there is no such thing as a private defect, that is, one detected and removed without being recorded.

Problem Addressed:

The only way to keep program costs from exploding is by finding and fixing defects as they occur. The cost of fixing defects typically increases by a factor of ten as they pass into each subsequent development phase.

Practice Essentials:

- Establish a goal for delivered defects per unit of size (defects include requirements problems)
- Implement practices to find defects when they occur
- Track average and maximum time to close a defect after it's reported
- Track defect removal efficiency of:
  - All defects found through all techniques reported to and tracked by CM
  - All defects reported from field for first year after deployment

- Grade developers on defect removal efficiency:

$$\frac{\text{Number of Defects Found \& Fixed During Development}}{\text{Number of Defects Found \& Fixed During Development \& First Year in Field}}$$

STATUS CHECKS:

- Are defect targets established for the project? Are the targets firm?
- Are consequences defined if a product fails to meet the target?
- Do project quality targets apply to all products?
- Are there circumstances defined under which quality targets are subject to revision?
- What techniques are used to project latent defect counts?
- How are current projected levels of defect removal empirically confirmed as adequate to achieve planned quality targets?
- Is test coverage sufficient to indicate that the latent-defect level achieved by the end of testing will be lower than the established quality targets?
- Are the inspection and test techniques employed during the program effective in meeting quality targets?
- Do all discovered defects undergo CM, and are accurate counts achieved for defects discovered and defects removed?
- Is there a closed-loop system linking defect actions from when defects are first detected to when they're resolved?
- Is defect information defined at a level of granularity that supports an objective assessment of resolution on a periodic basis?

The cost of removing defects typically accounts for between 40 percent to 50 percent of all development costs. The bigger software becomes, the greater the rate at which defects occur—the "software defect snowball effect."

Typically 20 percent of all softwar modules contain approximately 80 percent of the defects.

8. CONFIGURATION MANAGEMENT

Configuration Management is an integrated process for identifying, documenting, monitoring, evaluating,

controlling, and approving all changes made during the life cycle of the program for information that is shared by more than one individual. The discipline of CM is vital to the success of any software effort.

PROBLEM ADDRESSED:

Complexity is the cause of most of the problems attacked by the Best Practices. There is an inherent complexity in large-scale software projects that cannot be reduced. However, it is very easy to increase software development and maintenance complexity to a level that greatly exceeds this inherent complexity. Failure at Configuration Management is a sure way to dramatically increase complexity to the level of chaos.

PRACTICE ESSENTIALS:

- Formally track the status of problem reports, Engineering Change Proposals, etc.
- Control change to:
  - Deliverables
  - Cost/schedule baselines
  - External interfaces
- Have a formal process for making change to a baseline
- Automate the CM process when possible

STATUS CHECKS:

- Is the CM process integrated with the project plan, and is it an integral part of the culture?
- Are configuration control tools used for status accounting and configuration identification tracking?
- Are periodical reviews and audits in place to assess the effectiveness of the CM process?
- Are all pieces of information shared by two or more organizations placed under CM?
- Do you have a process to measure the cycle time?

MIL-STD-973 provides a good definition of Configuration Management.

9. PEOPLE-AWARE MANAGEMENT ACCOUNTABILITY

Management must be accountable for staffing qualified people (those with domain knowledge and similar experience in previous successful projects) as well as for fostering an environment conducive to high morale and low voluntary staff turnover.

PROBLEM ADDRESSED:

Perhaps the single most important determinant of project success is the quality, experience, and motivation of the people working on it.

No matter how well versed a software person is in the technology relative to the job, a substantial investment is required to bring that person to a level of detailed understanding about the application being developed or maintained. In spite of efforts to document the

software, vital information about your project exists only in the minds of select individuals. And with the rapid advances in software technology that have occurred and will continue to occur, a high percentage of software professionals are not proficient in the best technology related to their job. A significant part of software development and maintenance requires human intellect and creativity at a level exceeding that required for most jobs. Demands upon intellect and creativity are even greater with a number of the new technologies such as client/server networks and the abstractions required for object- oriented analysis and design.

PRACTICE ESSENTIALS:

- Ensure that one of the project manager's incentives is to maintain staff quality and a low voluntary turnover rate
- Keep records of actual hours worked as well as hours charged to the customer (Extended periods of actual work greatly in excess of 40 hours per week are an indicator of excessive future voluntary staff turnover.)
- Treat your stars well (There's a very high near-term and projected long-term demand far in excess of supply for computer engineers and computer systems analysts.)

STATUS CHECKS:

- Will the procuring/developing program manager be on board for the entire project?
- Are domain experts available?
- Does the project manager have software experience in a project of similar size and objective?
- Are all personnel fully aware of their role in the project?
- Is quality of performance acknowledged?
- Is personnel continuity ensured in light of changing company or program needs?
- Are opportunities for professional growth available to all members of the project?
- Do the developers believe in the goals of the project and that the schedule is feasible?
- Is the motivation and retention of personnel a key part of management assessment?

The preceding eight Principal Best Practices will be of little help if the technical staff is not qualified or quits.

Studies of large projects have shown that 90th percentile teams of software people typically outperform 15th percentile teams by factors of four to five, with individual productivity ranges of 26:1.

Studies have also shown a high correlation between organizations that invest in timely training and their development success, and that a high percentage of software professionals do not keep up with the technology

In the event that a program manager desires additional information, the Network will provide source materials and access to experts. We would greatly appreciate receiving your comments and suggestions (preferably by e-mail).

E-MAIL: BEST@SPMN.COM PHONE: (703) 521-5231 FAX: (703) 521-2603

PRINCIPAL BEST PRACTICES	CONTROL PANEL GAUGES													
	Cumulative Earned Value Delivered	Cumulative Months	Cumulative Dollars	CPI	TCPI	"Abba Chart"	Tasks	Quality Gates Progress	Aggregate Schedule Overrun	Aggregate Requirements Growth	Voluntary Turnover	Overtime Hours	Risk Impact	Risk Liability	Warnings	Defects by Activity
Formal Risk Management								X					X	X		
Agreement on Interfaces										X						
Formal Inspections																X
Metrics-based Scheduling and Management	X	X	X	X	X	X	X		X	X	X	X	X			
Binary Quality Gates at the Inch-Pebble Level																X
Program-wide Visibility of Progress vs. Plan															X	
Defect Tracking Against Quality Targets	X	X		X	X	X	X	X	X							
Configuration Management										X						
People-Aware Management Accountability									X		X	X			X	

FIGURE 5.1 RELATIONSHIP BETWEEN GAUGES AND PRINCIPAL BEST PRACTICES

CHAPTER 6

BEST PRACTICES

CHAPTER 6

BEST PRACTICES

These practices are derived from practices used by successful commercial and defense software projects.

Because the practices are not tied to a specific metric or method, program managers can apply a practice in response to particular corporate and program needs.

The practices are grouped into seven proven management areas (see Figure 6.1):

- Risk Management
- Planning
- Program Visibility
- Program Control
- Engineering Practices and Culture
- Process Improvement
- Solicitation and Contracting

Of course, many of these practices apply to more than one management area, though for the sake of simplicity they've only been listed once.

1. RISK MANAGEMENT

Risk management is vital to effectively managing any large-scale software effort because each software system being developed or maintained is dependent on a unique set of changing factors.

Risk management includes:

- Estimating the likelihood of identified risks occurring
- Establishing potential short-term and long-term consequences of risks
- Establishing a strategy and methods for comprehensive risk management
- Monitoring

Risks often encountered on software projects include, but are not limited to:

- Large system size
- Unclear and changing requirements
- New technologies
- System complexity
- Scope not adjusted to budget
- High dependency on specific people

Risks are incurred for reasons including:

- An inability to gauge the true extent and complexity of efforts to be accomplished
- An inability to accurately predict the extent of resources (material and time) necessary to complete efforts
- An organization tries to exceed its competencies
- User requirements are unstable
- The software environment and tools remain immature
- The technical difficulties are more difficult than anticipated
- Schedules are based on predetermined dates rather than quantitative estimates
- Attempting efforts in a manner inconsistent with the organization's culture
- Relying on silver bullets to induce large productivity improvement

Risk Management Best Practices include:

- Establish Management Reserves for Risk Resolution
- Implement Metrics-based Risk Decisions
- Perform Continuous Risk Management
- Formalize Risk Tracking and Review
- Manage Impact of External Dependencies

PRACTICE: ESTABLISH MANAGEMENT RESERVES FOR RISK RESOLUTION

Every project should have a reserve of money and potential staff marked for addressing risk and its potential effects.

PRACTICE ESSENTIALS:

Management reserves include:

- **Time—**Some experts recommend adding 10 percent above the estimated time-to-delivery to the schedule.
- **Money—**Potential additional staff, tools, and time add potential project costs.
- **Staff and potential staff—**The personnel organization should continue to interview for good people.

PROBLEM ADDRESSED:

Lack of an adequate risk reserve can stop fixed-rate contract completion and preclude adequate resources on a cost-plus contract.

REQUIREMENTS:

- Project risk analysis, with prioritized risks assigned an estimated value
- Government program office support for risk reserve in contractor bids

STATUS CHECKS:

- Is a risk reserve buffer established at the beginning of the project? Do fluctuations in the buffer occur only as a result of revised risk assessments?
- Are adequate risk alternatives documented to permit resource redistribution in the event of unexpected or expected risks being realized?
- Can expected risks be realized and shortfalls in project resources be recognized early enough to allow for correction?
- Are checks and balances built into the development process to continually evaluate resources and suggest alternative allocations to maintain project integrity?

Risk reserve reevaluations and updates should be performed along with risk projections and assessments.

Contractors (or developing agencies) should not be held responsible for the costs of creeping user requirements.

Throwing more programmers at an overlong project can make it even longer, but the right person in the right job can make a big difference in product results.

Cost-plus contracts without appropriate incentives tend to decrease motivation for contractors to establish risk reserves.

PRACTICE: IMPLEMENT METRICS-BASED RISK DECISIONS

Metrics selection should be identified up front and used to automatically trigger management reviews.

PRACTICE ESSENTIALS:

- Predefined limits (such as a module of code exceeding its estimated size by 10 percent) and specific conditions (such as not finding expert staff when needed) that automatically trigger management reviews
- Time series analysis to project and analyze long-term risks
- Metrics to measure risks concerning:
  - Cost
  - Effort or staffing
  - Schedule
  - System size
  - Successful activity completion
- Product defects in terms of:
  - Defect density
  - Defect removal rates
  - Incoming defects rate
  - Defect closure rate

PROBLEM ADDRESSED:

Schedule slippage, cost overruns, low quality, dissatisfied customers, and even program cancellation are potential results when established metrics are not used as risk indicators.

REQUIREMENTS:

- Metrics that accurately represent cost, product, quality, or organizational processes
- Measurements that trigger management reviews
- Experience in actual risk management and mitigation
- Management reviews ensure timely corrective actions for cost overruns, schedule slip, or performance deficiencies
- Documentation of decisions and actions taken

STATUS CHECKS:

- Does the risk management plan define reviews and triggers that must be quantitatively specified and monitored?

Guidelines to determine useful metrics for risk identification, analysis, and management include:

- *Characteristics of the software development process*
- *Maturity of the software development process*

(According to Grady, projects should invest about 18 percent of their total project effort during requirements specifications to double check.)

[1] R. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, 1993.

PRACTICE: PERFORM CONTINUOUS RISK MANAGEMENT

Enabling proactive risk management iteration throughout the project life cycle is the most efficient way to identify and mitigate risks before they become serious problems.

PRACTICE ESSENTIALS:

- Iterative assessments
- Identification of key program risks
- Prioritization
- Communication paths for user involvement

PROBLEM ADDRESSED:

Uncertainty is inherent in software projects because of lack of information, advances in technology, and system complexity.

REQUIREMENTS:

- A proactive approach to risks
- Plans for risk mitigation steps
- An organizational structure and activities for managing risk throughout the product life cycle
- Analysis
- Action planning
- Tracking and control
- Automated risk management tools

STATUS CHECKS:

- Is the risk database formally and routinely updated to reflect changes to risk status?
- Is a list of the top ten risks produced at the beginning of the project and updated at least monthly?
- Does evidence exist that identified risks have been mitigated or accepted?
- Does data for each risk include risk type, description, consequences, and potential mitigation approaches?

Use a risk identification checklist as a guide to prepare a list of potential risk items against the Work Breakdown Structure.

Risks that are "showstoppers"—with the potential for significantly impacting the project—should be reported on regularly

PRACTICE: FORMALIZE RISK TRACKING AND REVIEW

All projects should establish and maintain a formal risk tracking system that incorporates risk traceability.

PRACTICE ESSENTIALS:

- A documented risk management plan
- A documented risk management process
- A prioritized list of risks
- Meetings with customers about major risks

PROBLEM ADDRESSED:

Risks that are not traced or tracked are not managed, may be forgotten, or dismissed as minor, and yet they may have serious repercussions later in the program. Formal risk tracking and review increases the recorded knowledge concerning program risks, facilitates communication of risk status, provides traceability to the risk source, and ensures proper management attention to risk.

REQUIREMENTS:

- Regularly scheduled reviews by senior management
- Management attention to risk
- An automated tracking system that incorporates risk traceability
- A risk database that saves all entries in case of potentially related problems
- Formal presentations to management on risk topics, including importance of risk, severity of risk, how risk can be mitigated, and status of current high-priority risks

STATUS CHECKS:

- Are risk procedures documented?
- Does the staff understand how the risk tracking system works?
- Are risks assessed and prioritized both in terms of likelihood and potential impact?

Be aware of cultural resistance to formalizing risk management processes.

When risks are controlled, with a few exceptions, plans are executed during normal working hours.

The program manager has the ultimate responsibility for validating and tracking all risks during the definition of material outcome and the contract placement phases of the acquisition model.

PRACTICE: MANAGE IMPACT OF EXTERNAL DEPENDENCIES

All products that come from an external project or result from an external action should be identified and monitored.

PRACTICE ESSENTIALS:

- Monitor dependencies that have potential significant impact until the risk is eliminated, substantially decreased, or until the end of the program

PROBLEM ADDRESSED:

External dependencies, including activities, resources, information, conditions, and influences can potentially cripple a project.

REQUIREMENTS:

- Memoranda of understanding with organizations that control the external dependencies
- Contract to limit liability of risks resulting from external dependencies (for example, failure of the government to deliver the product on time)

STATUS CHECKS:

- Are external dependencies clearly identified?
- Are external dependencies represented on the Work Breakdown Structure?
- Are procedures in place for monitoring and managing the status of the external dependencies?

Because internal program problems demand immediate attention, too little attention is paid to external dependencies, which increases risk.

Contingency scenarios for identified external dependencies should be developed.

2. Planning

Software management planning includes:

- Defining the product goals
- Structuring the project
- Reviewing the plan
- Scheduling the project
- Testing the plan
- Costing the plan
- Frequently revising the plan as project circumstances change

Planning starts top down—a client needs software to solve a specific problem—and it should proceed mostly bottom up, as plans are refined or changed.

Compare your plans to previous experience and historical software project data.

At the beginning of the project, resolve standard definitions for:

- Valid product and process measurements
- Mutually acceptable analysis techniques
- User product acceptance criteria

Resources can be estimated in several ways:

- By analogy to similar projects
- According to expert studies and opinions
- Build-to-cost
- Top down
- Bottom up
- Cost modeling

After initial brainstorming sessions, team planning should continue as short, formal meetings followed by short, clear, well-distributed minutes.

Planning should be iterative and evolutionary, with plans updated regularly to reflect project realities.

The project plan should be clearly documented and made available to everyone on the project.

High- and low-level tasks should be described in terms of their:

- Purpose
- Performance approach
- Inputs
- Outputs
- Metrics
- Risks
- Resources

Plans should be tied to tasks and layered onto the project master schedule to cross-check budget and schedule adequacy.

Results of estimation tools should be corroborated or corrected with estimation verification tools.

BEST PRACTICES

During initial planning, a quality plan should be produced that identifies the quantitative goals for software product quality, metrics, and any process changes that are needed from the start.

Planning Best Practices include:

- Quantitative Software Estimation/ Verification
- Joint Team Involvement
- Activity Planning
- Data Requirements

PRACTICE: QUANTITATIVE SOFTWARE ESTIMATION/ VERIFICATION

Quantitative cost estimation and verification are essential to reducing estimation errors. Commercial estimating tools tuned to an organization's database of past projects increase the likelihood of accurate estimates.

PRACTICE ESSENTIALS:

- Software system size is the cornerstone estimate. Size affects the schedule, effort, cost, productivity, and quality of software systems. Software bugs, or defects, grow at a rate linear to system size.
- Large software systems consist of millions of lines of code, too much for an individual or small group to fully understand, much less write. Current methodologies for sizing systems are:
  - Counting Source Lines of Code (SLOC)
  - Estimating processing events that will take place for a specified duration
  - Assigning and adding up Function Points, a mathematical model in which Points correspond to features to be delivered to the client

PROBLEM ADDRESSED:

Inaccurate estimates are caused by ineffective estimating tools, lack of an accumulated historical database, and/or growth in project scope.

REQUIREMENTS:

- Software cost estimates predicated on the developer's past performance in the domain and on historical data
- Regularly scheduled estimate updates that detail changes to baselines and their rationale, including progress, quality, and cost
- Regular updates to correspond with reality throughout the project, and reflected in the contractor's software development plan or through other media established by the contract

STATUS CHECKS:

- Is more than one type of estimation technique used, and are the results compared and analyzed?
- Is historical data used to support the estimates?
- Are the estimation process, assumptions, risks, and results documented?
- Is the estimate reviewed with higher management and the customer?
- Are estimation techniques reapplied regularly to produce more accurate forecasts as new data becomes available?

All estimates should evolve along with the product under development.

Changes and updates to the contractor's proposal should be mutually agreed to by the acquirer and the contractor

Documenting the calculations of individual estimates allows for easier review and understanding.

PRACTICE: JOINT TEAM INVOLVEMENT

Multidisciplined support teams should translate initial and evolving user needs into clear, concise, and complete system requirements.

PRACTICE ESSENTIALS:

- Techniques that capitalize on customer/contractor involvement such as:
  - Integrated Product Teams (IPTs)
  - Joint Requirements Planning (JRP)
  - Joint Application Design (JAD)
  - Internal and user prototype demos, especially of the interface as it evolves
  - Structured acceptance test sessions

PROBLEM ADDRESSED:

Failure to meet customer and user expectations caused by missing or ineffective communication among stakeholders.

REQUIREMENTS:

- Active participation by team representatives
- Prewritten agenda and constraints for all team meetings
- Clearly stated goals and objectives
- Professional facilitation to assure focus, participation, and mutually understood communications
- Team members from the following organizations and domains of expertise:
  - Acquisition management
  - Users
  - Developers
  - Product support personnel

STATUS CHECKS:

- Is team involvement planned to ensure that requirements are understood by designers and developers?
- Are the project objectives well defined and documented?
- Does the team have experience in the application area and in the project approach?

Representatives must all be committed to the team and its objectives, or the team will not survive the product life cycle.

Moving project rework from the construction phase of a project to the design phase can be accomplished by JAD.

Structured user acceptance testing helps prevent errors before a product is fielded.

PRACTICE: ACTIVITY PLANNING

Detailed milestones used by the developer should be scheduled in evolutionary "inch-pebbles."*

PRACTICE ESSENTIALS:

- A top-level **Gantt chart** showing major project commitments is the timetable for product delivery to the customer
- A mid-level **PERT chart** shows milestones to the end of the project and relationships among project tasks
- Lowest-level activities represented as inch-pebbles, because:
  - Contingencies can be planned by estimating negative impacts on the sum of individual inch-pebbles
  - Impact of potential problems of resource availability or shortages can be analyzed to determine potential slips
- A detailed PERT chart at the task level, with each task in the activity network described in terms of duration, relationships, and dependence on other tasks being completed on schedule.
- Short-term or evolutionary planning that allows:
  - Identification of ineffective activities
  - Dynamic allocation and reallocation of resources
  - Responsiveness to changes in long-term goals

PROBLEM ADDRESSED:

Traditional planning creates inaccurate estimates and significant risks for large, complex projects.

REQUIREMENTS:

- Activities planned around major program milestones and design reviews
- Similar practices used by both acquirer and contractor teams to establish hierarchical reviews to support the different levels and associated needs of the acquirer and contractor organization
- Program management involved in the development process without micro-management
- Detailed program plans developed for all tasks including:
  - Task description

* See the footnoted AIA comment on p. 39.

- Approach
- Inputs/outputs
- Schedules
- Metrics
- Resources needed

- Hierarchical schedules that provide information that meets the needs of all personnel involved in the project, because the upper-level schedules are broad-based while the lower levels are specifically task-based
- Methods for monitoring and controlling potential slips for tasks with greatest schedule impact

STATUS CHECKS:

- Is there a hierarchical set of plans (or at least high-level and detailed plans) identifying roles and responsibilities, end products, life cycle stages, etc.?
- Are individuals associated with the accomplishment of specific activities?
- Are Gantt charts updated weekly?
- Are milestones pass/fail?

A project planning tool should be used when possible.

Early milestones should become more detailed as data becomes available. When possible, high-risk items should be identified and kept off the critical path through concurrent scheduling and risk mitigation activities.

PRACTICE: DATA REQUIREMENTS

Data requirements should address programmatic, process, and product needs, enable effective customer participation in product development, and support the product life cycle.

PRACTICE ESSENTIALS:

- Management, planning, engineering, control, and maintenance support of the project data
- Contractor's processes, methods, and environment data
- Contractor's appropriateness for product support data
- Training necessary to provide full team access to and usage of the newly developed software

PROBLEM ADDRESSED:

More data is often considered better than less data without realizing the cost of the data and the tradeoffs between producing data vs. putting more effort into developing better quality software.

REQUIREMENTS:

- Address programmatic, process, and product needs
- Enable effective customer participation in product development
- Support intelligent tradeoffs during the product life cycle based upon competitive prices and open to all prospective competent providers

STATUS CHECKS:

- Are requirements for shared data identified early and clearly?
- Are project plans scaled to software development requirements?

3. PROGRAM VISIBILITY

Visibility into product and project progress should be available and understandable to every member of the project team. To control and manage software projects, program managers need readily visible information that provides status and trends. Information-based software program management includes:

- Identification and definition of the program software issues and objectives
- Quantitative and qualitative data collection and analysis
- Evaluation of analysis results and possible courses of action
- Implementing corrective action

Visibility into the development processes can be gained by:

- Selecting software and program metrics that promote project visibility
- Attending software development meetings
- Sharing changes with contractor and customer program management

Measurement provides a software organization with a means to make progress visible. Accurate metrics allow project managers to:

- Measure specific areas of software product or process
- Derive basis for estimates
- Track project progress
- Monitor software quality
- Analyze defects
- Monitor process improvement

The major indicators of product status —the Control Panel indicators (Chapter 2)—should be:

- As accurate as possible
- Updated weekly
- Viewable by all project members

The Control Panel, preferably computer-generated, should be posted in high-traffic areas like the cafeteria and on the way to rest rooms.

Program Visibility Best Practices include:

- Practical Project-Oriented Software Measurement Process
- Issue-Driven Measures
- Internal Engineering Analysis Process
- Effective Communication Structure

PRACTICE: PRACTICAL, PROJECT-ORIENTED SOFTWARE MEASUREMENT PROCESS

An agreed-upon, documented measurement process with trained users, and backed by upper-management commitment, should be designed to track and communicate measures that promote visibility of the project.

PRACTICE ESSENTIALS:

- Software measures driven by program-specific issues and objectives
- Automated tools to process the data generated by product development, including:
  - Data collection
  - Data processing and management
  - Results analysis
  - Reporting

PROBLEM ADDRESSED:

The software measurement process provides the framework for structured and consistent identification and evaluation of software program objectives, status, and issues.

REQUIREMENTS:

- Project visibility
- Appropriate and accurate measurements
- Committed backing from upper management
- Agreement by acquirer and contractor
- Clear and current documentation
- Collection and analysis of low-level software data
- Senior management commitment in both the developer and acquirer organizations

STATUS CHECKS:

- Are measurement results widely used in organizational decision making, and are they communicated and accepted outside of the software development organization?
- Are project reports based on measurable data?
- Does project staff accept, and assist in, data collection?
- Do both the acquirer and developer have access to the software measurement data and maintain independent measurement analysis capabilities?

The acquirer, contractor, and customer should use the same models and metrics to measure the same system characteristics.

The software measurement data and subsequent analysis results must be considered in the context of other "engineering" information from the same program.

The software measurement process is applied to support program planning, development, and sustaining engineering.

Data from completed projects should be collected in a database to serve as input for future projects.

PRACTICE: ISSUE-DRIVEN MEASURES

Measures should be based upon answering a need, question, or issue in the software project.

Practice Essentials:

- A goal-driven metric paradigm
- Plans, deltas to plans, and actual measurements
- Predefined pass/fail criteria
- Modifications from the baseline per development activity
- Work Breakdown Structure design
- Metrics to:
  - Derive the basis for estimate
  - Track project progress
  - Verify quality levels
  - Analyze defects
  - Validate Best Practices

Problem Addressed:

Measures that are not driven by specific project needs appear to be trivial and, as a result, are not carried out.

Requirements:

- Timely data collection and processing
- Measurement data characterized as to source
- Flexible measures for changing issues and objectives

Status Checks:

- Are project turnover rates tracked?
- Are collected measures clearly linked to specific project needs?
- Are the number of on-time quality gates tracked?
- Are the number of unresolved high and low risks tracked?
- Is the incidence of requirements creep tracked?
- Are the number of defects opened and closed tracked?

The fundamental metrics to help program managers evaluate the project are verified progress against plans, costs, and quality levels.

PRACTICE: INTERNAL ENGINEERING ANALYSIS PROCESS

The quantitative and qualitative output of reviews such as peer reviews, inspections, and walkthroughs provides visibility of the project.

PRACTICE ESSENTIALS:

- Clearly defined entry and exit criteria for each activity
- Predefined, documented metrics
- A purpose focused on a well-defined scope of technical or management issues and objectives
- A common and well-defined documentation and reporting structure for implementing activities

PROBLEM ADDRESSED:

It is difficult to control a project if you can't assess its goals or progress toward them. The results of inspections and reviews provide significant visibility into the status of the project.

REQUIREMENTS:

- Measurement program to capture data
- Use of reviews and inspections
- Project structure that ensures data collection feeds the measurement process
- Output of reviews and inspections is tracked and reported for collection
- Identified problems from reviews and inspections are placed under configuration control to ensure completion

STATUS CHECKS:

- Are metrics used to gauge the effectiveness of formal inspections, walkthroughs, and reviews?
- Are the results of inspections and walkthroughs used to estimate or predict the level of quality in the product?
- Is the output of reviews and inspections tracked and reported for collection?
- Are problems that are identified in reviews and inspections placed under configuration control to ensure completion?
- Are inspections conducted by qualified staff sufficiently knowledgeable to assess quality?

Data analysis must have a clear goal in order to obtain the desired information, not necessarily the desired results.

PRACTICE: EFFECTIVE COMMUNICATION STRUCTURE

Effective communication means open communication that gives the development team, management, and the client easy access to project status and information.

PRACTICE ESSENTIALS:

- Technical Interchange Meetings (TIMs)
- Program Management Reviews (PMRs)
- Management Status Reviews (MSRs)
- Technical Working Group Meetings (TWGMs)
- Event-driven baseline/milestone reviews
- Independent Verification and Validation (IV&V) processes

- Software product and process quality and compliance audits
- Software product prototypes and demonstrations
- Groupware, or collaboration software, to include:
  - On-line meeting arrangers
  - On-line bulletin board services, e-mail lists for memos, minutes, and messages
  - Desktop publishing with automated templates for software engineers who do not like to write
  - An on-line reference library with divisions for technical, business, and marketing articles, or references to their whereabouts

PROBLEM ADDRESSED:

As the project grows, an effective communication structure must keep pace with the need for information flow.

REQUIREMENTS:

- An organization in which all staff understand their assigned roles, job responsibilities, and reporting commitments
- Meetings that act as tools for project reporting
- An automated reporting system that indicates status of the project and the basis for those indications

STATUS CHECKS:

- Are meetings held regularly and documented concisely?
- Have the acquirers and the developers agreed on current requirements?
- Are project members up to date with project and product goals?
- Are members of the project, top to bottom, plugged into an electronic communication network?

4. PROGRAM CONTROL

Software management program control requires:

- Planning
- Executing to the current plan
- Incorporating changes to the plan and project
- Meeting product resource and quality goals
- Managing change control
- Coordinating group and individual efforts
- Ensuring adequate quality checks to promote high software quality

Program control activities underlie the software development process, affect software quality, and include:

- Quality assurance
- Testing and evaluation
- Configuration Management

Quality gates at key points in the development process are used to monitor and ensure the quality and integrity of products. Quality gates include:

- Peer reviews
- Inspections
- Walkthroughs
- Structured project audits

Poor program control shows up in low quality, low productivity, low user satisfaction, cost overruns, long schedules or missed milestones.

Program Control Best Practices include:

- Test Methodology
- Regression Testing
- Computer-Aided Software Testing
- Error Source Location
- Independent Verification and Validation (IV&V)
- Quality Gate Completion Criteria
- Configuration Management Coverage
- Requirements Change Management
- Baseline Methodology
- Technical Quality Assurance

PRACTICE: TEST METHODOLOGY

A test approach that is tailored to, and consistent with, the development methodologies provides a traceable and structured approach to verifying requirements and quantifiable performance.

PRACTICE ESSENTIALS:

- Assurance that requirements and criteria are testable through design, coding, and test
- User, tester, and requirements developer involvement
- Consistency throughout the project
- Test methodology reflected in:
  - Task descriptions
  - Test plan
  - Data collection plan
  - Analysis methodology
  - Reports
  - Test requirements correlation matrix
  - Other development methodologies

PROBLEM ADDRESSED:

Test problems are caused by a lack of testable criteria, by tests that cannot be structured only as a final check, and by inconsistent testing.

REQUIREMENTS:

- A test methodology tailored to the project size and needs
- Professional testing/quality assurance staff
- A definitive strategy for software testing as part of the organization's framework

STATUS CHECKS:

- Is the test methodology agreed to by users, testers, and requirements developers?
- Is test coverage adequate for risk handling?
- Are test methodology meetings held regularly, and documented clearly and briefly?
- Are tests linked to, or traced to, requirements?

Test methodologies should be consistent with the set of development methodologies so that information can serve as data for other automated systems.

PRACTICE: REGRESSION TESTING

Regression testing must be conducted to find any new defects following test correction.

Practice Essentials:

- A set of test cases that will ensure problems have been fixed
- No new defects introduced when fixing old defects

Problem Addressed:

Inadequate regression testing results in damage to previously operable functions and capabilities.

Requirements:

- Computer-aided regression tools that:
  - Automate the execution, management, and verification of the test suites
  - Can capture and play back all events that occur during user sessions
- Libraries of error-free, effective, reusable regression test cases for projects with multiple releases, placed under technical and management control

Status Checks:

- How does regression testing ensure that defect removal was successful and no new defects were created?
- How are test case suites for regression testing determined?
- What procedures ensure that the appropriate amount of regression testing is performed?

All software changes that have an increased likelihood of secondary failure, fault error introduction, including assembly languages, patches and conditional compilations, need retesting.

PRACTICE: COMPUTER-AIDED SOFTWARE TESTING

Computer-aided software testing combines proven testing design, development, and management practices with advanced digitized testing tools.

Practice Essentials:

- A test plan that is compatible with the development schedule
- Clearly defined methodology for assessment and evaluation
- Test coverage for the entire project
- Quantification of functional and technical project risks
- Definition of the test population
- Definition of testing metrics
- Repeatable testing practices that reduce the subjective nature of results and shorten retest time

BEST PRACTICES

PROBLEM ADDRESSED:

Testing software is very time- and labor-intensive.

REQUIREMENTS:

- Knowledgeable personnel
- Investment in digitized testing tools
- Tools for measuring defect potential

STATUS CHECKS:

- Are tools used to automate testing where possible?
- Is there adequate evidence that automated tools test what they are designed to test?

Computers will test only what they are instructed to test, and will analyze results according to the selected algorithm.

Inadequate use of automated tools for software quality checks is associated with a number of software risks such as cost overrun and missed schedules.

PRACTICE: ERROR SOURCE LOCATION

Techniques for identifying the source of errors should be utilized.

PRACTICE ESSENTIALS:

- Designing the system for testability
- Designing a comprehensive test plan, including:
  - What to test (you cannot test everything)
  - How to recognize an error
  - How to track tests results
  - How to deal with software changes
  - How to deal with test plan changes
  - Configuration Management version control
  - Error tracing to the module level
- Designing a test suite that includes:
  - Repeatable test cases
  - Kiviat charts for quality thresholds
  - Metric analyzers to focus on complex or undertested modules
  - Regression tests for software changes
  - Cyclic or random testing
  - Stress-load testing
  - Defect causal analysis
  - Path coverage for critical modules

Problem Addressed:

Unless techniques provide accurate information about error source locations, it is not possible to make the correct fixes.

Requirements:

- Multiple discovery techniques, including inspections, reviews, tests, automated error-locator tool and training in its use

Status Checks:

- Can errors be tracked from discovery to disposal?
- Are errors prioritized according to user requirements?

PRACTICE: INDEPENDENT VERIFICATION AND VALIDATION (IV&V)

Objective, unbiased software verification and validation should be conducted by an independent agent.

Practice Essentials:

- Analysis of program requirements
- Analysis of program design
- Analysis of program code
- Program testing
- Development and/or use of automated tools
- Technical evaluation

Problem Addressed:

- Late identification of software errors
- Poor software quality
- Lack of management visibility into the development process
- Inability to control schedule slips and cost overruns
- Lack of effective risk management methodology
- Inadequate systems integration planning

Requirements:

- Independent testing and evaluation of the technical acceptability of the software in terms of:
  - Operational requirements
  - Readiness of software system for its intended use
  - Integrity of the completed system for its mission
- An independent set of test tools and simulations
- An objective assessment of the correctness of the development team's solution at each phase of design
- Independent tests, tools, and simulations to confirm performance of the software against the specified performance

- Comparisons with development organization of errors vs. time

STATUS CHECKS:

- Are test specifications, designs, cases, and reports produced?
- Are metrics on defect data analyzed to enable product improvements?

If necessary, verification and validation can be functionally independent, organizationally independent, or independent through program control.

PRACTICE: QUALITY GATE COMPLETION CRITERIA

For each quality gate identified, criteria should be developed that indicate successful completion of the gate.

PRACTICE ESSENTIALS:

- Quality gate completion criteria for inspections, walkthroughs, reviews, tests, requirements, design, code, and documentation
- Process standards
- Product standards
- Historical data
- Limits
- Corrective action requests promptly answered
- Real-time feedback on development process adherence

PROBLEM ADDRESSED:

Without well-defined completion criteria, quality assurance is forced to catch up with defect detection during later phases of the program, which is inefficient and costly.

REQUIREMENTS:

- User-developer teams to agree on completion quality
- Support from upper management

STATUS CHECKS:

- Are there evaluation standards for all engineering data products at each quality gate?
- Has a quality-level threshold been established that all products must meet or absolutely be repeated?
- Is a set of quality inspections in place to assess the products and process attributes of the software, and is it effective?
- Are unambiguous quality criteria or predefined performance standards established for each product or deliverable?

- Are quality gates integrated into the project such that they check product quality at discrete points in the project infrastructure just prior to general use?

Samples should be taken randomly from the design, code, and documentation to determine whether the product passed the last gate with an excessive amount of defects. If so, the product should be returned to the previous process.

Criteria should be measurable, simple, and pass/fail.

PRACTICE: CONFIGURATION MANAGEMENT COVERAGE

Configuration Management (CM) applied effectively throughout the program prevents uncontrolled, uncoordinated changes to shared project information.

PRACTICE ESSENTIALS:

- Identification of all data
- Documentation internal and external to the project
- Monitoring standards and procedures
- Evaluation of the status and consistency of all shared project information

PROBLEM ADDRESSED:

Lack of effective change control increases the number of defects and amount of rework required during the project.

REQUIREMENTS:

- CM applied to requirements, specifications, design documents, and code, including:
  - Version identifiers
  - Delta identifiers
  - Derivation records
- Automated problem tracking and CM tools to help maintain accurate records of:
  - Proposed changes
  - Ownership
  - Test results
  - Implemented changes
- Three boards required:
  - Requirements Review Board
  - Engineering Review Board
  - Configuration Control Board or Change Review Board

STATUS CHECKS:

- Are review boards defining and enforcing CM procedures?
- Is it possible to trace a defect from first report to final disposal?

Few people like the administration of CM, but all must cooperate to make CM work.

PRACTICE: REQUIREMENTS CHANGE MANAGEMENT

A requirements management process supports the definition, identification, allocation, management, and control of all project requirements. Requirements must be satisfied and proposed changes evaluated for cost, schedule, and effects.

PRACTICE ESSENTIALS:

- Sufficient understanding of the need for control by team members and customers
- Rigorous configuration control of requirements to the lowest possible level
- Appropriate mapping of requirements to Configuration Items (CIs) and development activities
- A process that manages and controls requirements analysis, definition, tracing, and maintenance

PROBLEM ADDRESSED:

Loosely controlled requirements cause budget overruns and schedule delays.

REQUIREMENTS:

- Experienced personnel
- Computer support tools to maintain the requirements and to check for consistency

STATUS CHECKS:

- Are baselines updated as changes are made?
- Are project management and control procedures fully documented and well integrated into the development life cycle?
- Do requirements undergo formal change control, including prioritization of proposed changes, authorization for changes, and issue control of requirements documents?

Over the course of development, users' needs change.

To minimize damage to the softwar and schedule, select effective methods, tools, and approaches for managing requirements growth.

PRACTICE: BASELINE METHODOLOGY

Prior to beginning system definition, a methodology to establish a program baseline should be agreed to, approved, published, and followed.

PRACTICE ESSENTIALS:

- A user and supplier agreed-upon Initial Operation Capability (IOC) description
- Clear definitions of the program baselines throughout the project, and how each is to be developed, approved, recorded, changed, and measured

- Methods based on project requirements
- Consistency with other project tools
- Criteria to evaluate effectiveness
- Established as part of the culture

PROBLEM ADDRESSED:

Unrealistic program baselines drive project schedules, costs, and performance requirements astray.

REQUIREMENTS:

- Compatible with Configuration Management software
- Domain experts
- Product baseline, which describes the software in terms of function, performance, and operation
- Functional baseline, which is established after a successful requirements review
- Allocated baselines, which are established at the end of preliminary, or architectural, design

STATUS CHECKS:

- Have formal baselines been established after formal reviews?
- Are formal hand-off procedures established for CM baselines and responsibilities?
- Are standards in place for documentation included in the formal baseline?

Valid program baselines cannot be established without an understanding of the associated software on the project and the user and acceptance requirements.

PRACTICE: TECHNICAL QUALITY ASSURANCE

Projects need technical quality assurance, not format-checking quality assurance.

PRACTICE ESSENTIALS:

- People who are assigned quality assurance jobs in alternation with the sexier design jobs, and who are treated as value-added members of the development team

PROBLEM ADDRESSED:

Too often, quality assurance personnel on DoD projects serve as format checkers rather than performing a real technical role.

REQUIREMENTS:

- Organizational freedom but, as provided in MIL-STD-498, not necessarily organizational independence that historically leads to adversarial organizations
- Well-trained and well-paid personnel
- Government recognition of the need for quality assurance role change

STATUS CHECKS:

- Are reliability models equipped with defect tracking to predict the latent defects in the software?
- Are lower bounds defined for defect detection efficiency?
- Have conditional acceptance criteria been defined?
- Are the planned objective quality levels for the software consistent with the planned usage or mission of the software?
- Are procedures in place to assess the quality of deliverables before they are placed under configuration control?

Quality assurance should be able to show one-to-one correspondence between organizational activities and deliverable/product quality improvement.

Active quality assurance teams typically require about five percent of the staff of the group they support. Tasks performed include:

- *Predicting defect potential*
- *Predicting defect removal efficiency*
- *Predicting quality levels of software*
- *Measuring defect numbers, severity, etc.*
- *Assisting in selecting defect removal techniques*
- *Performing tests and validating results performed by others*
- *Measuring and calibrating defect removal efficiency*

5. ENGINEERING PRACTICES AND CULTURE

Good software engineering is both an art and a craft. Software engineering art makes software that solves a problem better than a previous way. Software engineering craft evolves continuously as programming techniques move from structured to object-oriented models, iterative phases complicate the development life cycle model, and new technologies are built and used.

Approaches to software development include:

- Phased design in which new tasks don't begin until prior tasks finish
- The release, or version, approach that delivers a product in which each additional increment is a semi-independent program
- The evolutionary approach, that takes advantage of the fact that, as more software is written, more and better ways to write software evolve. The evolutionary approach allows software coding to begin with a minimal number of firm requirements, and includes:
  - User orientation
  - More, faster iterations of analysis, design, build, test, fix, and retest cycles
  - Stress on definition and measurement of objectives
  - Multiple system attributes
  - Use of an existing system to start

Software development engineering needs include:

- Better programmer productivity
- Lower software maintenance costs
- Personnel to use new programming capabilities
- CASE tools in production environments
- Trained programmers, technicians, and users
- Automated design and test systems, including:
  - Interface builders
  - Software development workbench
  - Configuration Management system
  - Defect tracking system
  - Dynamic modeling and simulation
  - Integrated Computer-Aided Software Engineering (I-CASE)
  - Graphical User Interfaces (GUIs)
  - Interoperability with other software
  - Software portability across platforms
  - Distributed processing

Engineering Practices and Culture Best Practices include:

- Include User in a Multidisciplined Requirements Support Team
- Encourage Compatible Analysis and Design Methods
- Encourage Software Architecture Definition and Maintenance
- Encourage Requirements Engineering Process that Includes Use of Prototypes, Models, and Simulations

CHAPTER 6

BEST PRACTICES

- Encourage Proactive Change Impact Analysis
- Plan for Domain Engineering in Acquisitions
- Encourage Use of Clean Room Techniques
- Tailor Engineering Practices to Projects
- Encourage Use of Software Development Standards such as MIL-STD-498
- Assess Organizational Effectiveness

PRACTICE: INCLUDE USER IN A MULTIDISCIPLINED REQUIREMENTS SUPPORT TEAM

A multidisciplined requirements definition support team should translate initial and evolving user needs into clear, concise, and complete system requirements.

Practice Essentials:

- Team members from the following organizations and domains of expertise:
  - User command
  - Procuring organization
  - Support organization
  - Knowledgeable users, particularly with domain knowledge of the proposed system
  - Testers
  - Contractors
  - Developers after contract award

Problem Addressed:

Inaccurate or incomplete requirements definition is often the result of insufficient user understanding and contribution.

Requirements:

- User involvement through requirements evolution
- Goal to make sure that what is delivered meets the user's needs
- Service in an advisory capacity to the program office
- A process to assure consolidated user inputs to the team

Status Checks:

- Is user involvement planned to help ensure that designer's and developer's requirements are understood by staff?
- Are project objectives and customer restraints well defined and documented?
- Are all relevant stakeholders in the requirements document considered (designers, testers, customers, technical authors, management, and marketing)?
- Are customers educated in the limitations of computer solutions?

PRACTICE: ENCOURAGE COMPATIBLE ANALYSIS AND DESIGN METHODS

Analysis and design methods should be based on the project's domain area, application area, and desired output design model.

Practice Essentials:

- A model with the following characteristics:
  - Resilient to change
  - Extensible
  - Maintainable
  - Reliable
  - Verifiable

PROBLEM ADDRESSED:

Methods often don't take into account the particularities of the project and product.

REQUIREMENTS:

- Methods suitable to the domain area (signal processing, combat direction systems, communications systems, information systems, etc.)
- Methods suitable to the application area (real-time distributed processing, concurrent processing, etc.)
- Methods that enhance clarity of the desired software architecture
- Design model software components that closely resemble the entities of the problem domain
- Easy-to-understand design model
- Supported by an Integrated Computer-Aided Software Engineering (I-CASE) system

STATUS CHECKS:

- Are the development teams trained in selected analysis and design methods?
- Are the methods for different life cycle activities compatible?
- Are the methods adequate for the specific domain and application area?
- How are system and software architectural goals supported?

Methods support clear transitions from one development activity to the next.

PRACTICE: ENCOURAGE SOFTWARE ARCHITECTURE DEFINITION AND MAINTENANCE

A software-intensive system requires early definition of the software architecture that is consistent with operational scenarios and maintenance throughout development.

PRACTICE ESSENTIALS:

- Standards-based design built over a standards-based, service-layer model such as the Technical Architecture Framework for Information Management (TAFIM)
- Clear definition of the software architecture, preferably executable
- Software system functionality mapped to software system components
- Clear definition of relationships among components (interfaces, protocols)

- Rules for component composition (constraints), execution model (data and control flows), and clock
- User-operational scenarios
- Compatibility with currently operational systems

PROBLEM ADDRESSED:

Systems without clear and validated architecture lack flexibility to readily adapt to changing requirements and take advantage of reuse/COTS.

REQUIREMENTS:

- Rationale for design decisions
- Validation of the architecture by peer and expert review, including function, structure, and behavior
- Discussion of stability of the architecture through vendor updates
- A process for maintaining currency in architecture documentation

STATUS CHECKS:

- Is the software system architecture required as part of the response to the RFP?
- Is the software system architecture consistent with domain or product-line architectures?
- Are there plans to validate the software system architecture with the users?
- Is there a process in place to manage changes to the software system architecture and design during implementation?

It is reasonable to require bidders to specify the software system architecture and its maintenance in their proposals.

System architecture should be defined and approved early, and updated regularly to reflect changing conditions.

Deviations from the standard should be approved at the correct level, documented, and communicated to the domain or product-line manager as a recommended change.

PRACTICE: ENCOURAGE REQUIREMENTS ENGINEERING PROCESS THAT INCLUDES USE OF PROTOTYPES, MODELS, AND SIMULATIONS

An adequate requirements engineering process includes the use of prototyping, modeling, and simulation to define and clarify user requirements, and to validate implementation practicality.

PRACTICE ESSENTIALS:

- Rapid development of representation of a system characteristic including:
  - Physical structure of the system
  - Visible system capabilities offered by external interfaces
  - Internal constituent capabilities necessary to perform the required operations

  - System behavior
  - Conditions under which the functionality is offered or denied

- Modeling to guide the product development by synchronizing and focusing parallel development teams on simulating product reaction against real-world conditions, validating product requirements, and simulating product functionality

PROBLEM ADDRESSED:

Project cost overruns, schedule slippage, and compromised quality can be attributed to immature or improperly defined user and system requirements.

REQUIREMENTS:

- Clearly documented user requirements
- Validated product implementation plans
- Prototyping and modeling tools
- Updates and refinement over time

STATUS CHECKS:

- Is a customer representative and user group team defined with authority to make specific requirements decisions?
- Are there adequate provisions ensuring that the requirements are implemented in the design?
- Have functional and performance requirements been captured in a technically precise way?
- Are requirements reviewed for accuracy, consistency, and completeness?
- Are the system qualities like safety, security, performance, usability, learning requirements, and portability fully expressed?
- Are requirements prioritized, and is the rationale for the prioritization documented?

Prototypes do not eliminate the need for formal metrics and inspections on the project.

A prototype can deliver key components of the system, but is seldom more than 20 percent of the completed system; that's where the simulation comes in.

Prototyping has traditionally been a part of software development but, under evolutionary development, prototyping becomes an essential initial activity rather than throw-away demonstration.

Prototypes can help designers find the difficult parts of the system sooner rather than later

A weighted average of multiple models produces an estimate that more precisely represents the project than the results of any one model.

PRACTICE: ENCOURAGE PROACTIVE CHANGE IMPACT ANALYSIS

Change impact analysis should be proactive rather than reactive.

PRACTICE ESSENTIALS:

- A change implementation strategy, including testing metrics and validation criteria
- Processes to cover changes in threats, requirements, functionality, algorithms, interfaces, and hardware
- Externally mandated and internally generated changes

PROBLEM ADDRESSED:

The developer needs encouragement to implement change management, including prediction, impact analysis, planning, and results tracking.

REQUIREMENTS:

- Plans for implementation of an approved change within a systems engineering discipline.

STATUS CHECKS:

- Are procedures in place to assess the operational, engineering, and product impacts of all planned changes before they are made?
- Is the impact to software of proposed system-level changes routinely addressed?

Metrics serve to help make decisions about probable and possible changes.

PRACTICE: PLAN FOR DOMAIN ENGINEERING IN ACQUISITIONS

Domain engineering expertise is a major consideration in acquisition strategy options.

Domain engineering is a complex process of analyzing and modeling a domain, designing and modeling a generic solution architecture for a product line within that domain, implementing and leveraging reusable components of that architecture, and maintaining and updating the domain, architecture, and implementation models.

PRACTICE ESSENTIALS:

- Domain analysis, including:
  - An underlying theory and model
  - Analyzing a domain according to the model and reuse of systems
  - Use of development histories as data for management systems
  - A set of work products that approximates a specific domain or adapts to a particular organization
- Domain design, including:
  - A Domain-Specific Software Architecture (DSSA) that specifies components,

interfaces, and rules to compose systems

- Rationale for component selections

- Domain implementation of reusable software components that will fit the DSSA, use of components built for another system build on DSSA, and ensuring the reusability of those components

- Domain maintenance, including: correcting and enhancing domain assets (model, DSSA, and reusable components), and experience (positive or negative) with domain assets for support systems development

Problem Addressed:

Lack of an agreed-upon basis for developing sets of related systems that incorporate systematic reuse of COTS and other software assets causes excessive cost, schedule delay, and poor quality.

Requirements:

- Stable product line for valid domain engineering use

- Sufficient number of systems built in the domain to justify development of domain assets

- Access to personnel to provide domain expertise

Status Checks:

- How is the domain-specific approach to system acquisition being pursued?

- Has return-on-investment analysis been performed on candidate domain engineering investments?

- Is funding available that could be targeted to domain engineering activities, either before, as a part of, or concurrent with, one or more specific system acquisitions?

PRACTICE: ENCOURAGE USE OF CLEAN ROOM TECHNIQUES

The Clean Room process provides a rigorous engineering discipline within which software teams can plan, specify, measure, design, code, test, and certify software.

Practice Essentials:

- A specific set of managerial and technical practices for developing ultra-high-quality software with certifiable reliability

- An engineering discipline in which software developer teams:

  - Apply rigorous mathematical notation to specify, plan, develop, and verify software

- Utilize statistical quality assurance for defect prevention

PROBLEM ADDRESSED:

An undisciplined project environment can impede the development of high-quality software.

REQUIREMENTS:

- Formal specification and design that rely on disciplined engineering practices
- Software reliability engineering to measure software reliability and enforce process improvement
- Formal verification that compares specifications with the operating software
- Engineering activities separated into specification team, development team, and certification team

STATUS CHECKS:

- Have Clean Room practices been considered for adoption on the project?
- Has a trade study been conducted to evaluate and document the advantages and disadvantages of Clean Room application?

The benefits of Clean Room techniques have thus far only been realized in limited areas of software development. Clean Room techniques have not been proven in such areas as:

- *MIS projects*
- *Object-oriented analysis and design*
- *Client-server applications*

PRACTICE: ENTERPRISE PRACTICES TAILORED TO PROJECTS

The positive results achieved by commercial software development can be altered to meet the needs of government programs. Procedures and tools must have a supporting infrastructure within the organization.

PRACTICE ESSENTIALS:

- A library of accepted and tested Best Practices
- The organization's view of how to develop and/or acquire software

PROBLEM ADDRESSED:

Too many software-intensive systems are developed on an ad hoc basis, causing performance failures, missed schedules, and budget overruns.

REQUIREMENTS:

- Knowledgeable group of software experts available to tailor policies, practices, and procedures
- Rigorous adherence to Best Practices by project members with encouragement from the project manager

STATUS CHECKS:

- Does the organization identify and promote its own development Best Practices?
- Are those organizational practices specified in a manner to facilitate project adaptation?
- Are software experts available to support the adaptation of those practices?

PRACTICE: ENCOURAGE USE OF SOFTWARE DEVELOPMENT STANDARDS SUCH AS MIL-STD-498

Standards are essential to the stability of organizations and processes.

PRACTICE ESSENTIALS:

- Detailed requirements for:
  - Safety, security, and privacy
  - Project planning and management
  - Development environment
  - System requirements analysis
  - System design
  - Software requirements analysis
  - Software design
  - Software coding and unit testing
  - Software/hardware integration and testing
  - System acceptance testing
  - Software CM
  - Software product assurance
  - Software quality assurance
  - Risk management

PROBLEM ADDRESSED:

When many people, disciplines, and tools coexist on a large project, everyone must learn and follow a common way of doing the same tasks.

REQUIREMENTS:

- The product life cycle definition
- Software development processes
- Reusable software components

STATUS CHECKS:

- Are documents clear at the program management level?
- Are documents constructed from templates?
- Is standards definition under the control of an experienced expert?

Discipline is required to write, review, and carry out the plan for a complex software product and, to that end, MIL-STD-498 is better than previous standards.

PRACTICE: ASSESSED ORGANIZATIONAL EFFECTIVENESS

Key criteria to evaluate both the acquisition organization and the contractor organization for organizational effectiveness should be established.

PRACTICE ESSENTIALS:

- Criteria for assessing the acquisition organization's effectiveness, including:
  - Commitment and acceptance of responsibility
  - Accountability
  - Simple hierarchy that allows the project manager to address one chain of authority, to influence it, and to obtain recourse if needed
  - Technical support
  - Mechanisms for process improvement
- A model for assessing organizational maturity such as Software Development Capability Evaluation (SDCE), AFMC Pamphlet 63-103, SEI
- Criteria for assessing the contractor's organizational effectiveness, including:
  - After contract award, a customer/contractor Integrated Product Team (IPT)
  - Evaluation instruments such as the SEI Software Capabilities Evaluation (SCE) and the AFMC Pamphlet 63-103
  - Evaluations performed by someone with experience in the specific application area
  - Reevaluations performed throughout the development project

PROBLEM ADDRESSED:

Risk reduction is achieved by selection of a capable offeror, and early and continued visibility into capabilities.

REQUIREMENTS:

- A fully capable offeror with the capacity to develop software consistent with the Request for Proposal (RFP) requirements
- Early, comprehensive visibility into the offeror's proposed capabilities
- Continued visibility into the developer's actual implementation after contract award
- Continual productive communication between the program manager and the contracting officer

STATUS CHECKS:

- Does the development process provide smooth transitions of personnel responsibility and assessment of personnel adequacy as deliverables move through the development process?
- Are engineering checks and balances in place to identify personnel shortfalls before productivity is adversely affected?

Organizational effectiveness in knowledge-intensive software development organizations depends on teams of individuals.

6. PROCESS IMPROVEMENT BEST PRACTICES

Process improvement is in itself a process that:

- Starts by identifying the strengths and weaknesses in an organization
- Analyzes the options to capitalize on strengths and improve weaknesses
- Plans a process improvement process
- Monitors and reports on results by improvement plans

Process Improvement Best Practices include:

- Identifying and Fostering Sponsorship
- Establishing and Maintaining the Framework for Process Improvement
- Assessing and Reassessing an Organization's Process Capability
- Developing a Software Process Improvement Plan
- Institutionalizing the Software Process Improvement Plan
- Closing the Loop for Software Process Improvement

PRACTICE: IDENTIFYING AND FOSTERING SPONSORSHIP

Project sponsorship ensures successful long-term process improvement activities.

PRACTICE ESSENTIALS:

- Specific tasks that sponsors must perform to ensure that the initiative has appropriate visibility throughout the organization
- Communication of the business reasons for process improvement throughout the organization, one-on-one, in small groups, by department, and company-wide
- Cascading sponsorship through commitment expressed at all organization levels

PROBLEM ADDRESSED:

The commitment to sponsorship of software process improvement by program and product decision makers is often lacking.

REQUIREMENTS:

- Goal setting at every level of the organization
- Scheduled process improvements inspection including progress reviews, assessments, and evidence of improvement
- Resources that include but are not limited to: time, people, dollars, and equipment
- Personnel systems that recognize and reward behaviors beneficial to the project

STATUS CHECKS:

- Does top management act on feedback about project conditions?
- Do team leaders act as sponsors for their teams?

Groups, not individuals, are better recipients to reward, as championship teamwork is the goal.

Sponsorship should be rewarded.

PRACTICE: ESTABLISHING AND MAINTAINING THE FRAMEWORK FOR PROCESS IMPROVEMENT

A software process improvement framework should be established and maintained for all software acquisition.

PRACTICE ESSENTIALS:

- Adoption of a model or framework of tailorable templates for standards, operating procedures, techniques, tools, and education/training

PROBLEM ADDRESSED:

If an infrastructure for software process improvement is not established within an organization, software process improvement will not become part of the engineering group culture.

REQUIREMENTS:

- Organizational support for a Software Engineering Process Group (SEPG) to serve as a focal point for software process improvement initiatives
- Official recognition of software process improvement efforts
- A source of measurement and feedback
- A provider of training

STATUS CHECKS:

- Is the framework established and communicated to individuals on the project?

Process improvement in response to a problem is too late.

PRACTICE: ASSESSING AND REASSESSING AN ORGANIZATION'S PROCESS CAPABILITY

Organizations must establish a specific repeatable means to assess the strengths and weaknesses of their software development processes.

PRACTICE ESSENTIALS:

- The reason for assessment (for example, source selection, the start of a major process improvement effort, periodic evaluation)
- Quality models against which the software development processes will be evaluated (for example, ISO 9000, SEI's CMM, SDCE)
- Methods based on available resources and their allocation (for example, CBA-IPI is more expensive and detailed than SPA, which is more expensive than a mini-SPA)
- Bench-marking criteria to evaluate the organization's processes
- Determination of progress against the process improvement action plan and established relative to the quality model

PROBLEM ADDRESSED:

Organizations without repeatable means of identifying current capabilities cannot identify and improve problems.

REQUIREMENTS:

- Management's visible commitment to act on the findings of the assessment
- Comparison of software development practices with

practices of other organizations in order to identify and adopt the best-in-class practices

STATUS CHECKS:

- Are periodic assessments of process capability conducted?
- Are strengths and weaknesses listed?

PRACTICE: DEVELOPING A SOFTWARE PROCESS IMPROVEMENT PLAN (SPIP)

A Process Improvement Plan (PIP) should support business objectives, identify organizational strengths, and improve organizational weaknesses.

PRACTICE ESSENTIALS:

- Goals based on business objectives
- Process strength and weakness assessment
- Continued improvements over time

PROBLEM ADDRESSED:

Lack of a carefully thought-out and clearly documented SPIP results in false starts and conflicting activities.

REQUIREMENTS:

- Development reviews
- Active senior executive sponsorship
- Short-term goals to highlight early successes
- Visible reviews of periodic progress

STATUS CHECKS:

- Is the business need for the system or enhancement clear to project personnel?

Plan implementation should be tested in a pilot program.

PRACTICE: INSTITUTIONALIZING THE SOFTWARE PROCESS IMPROVEMENT PLAN

To be effective a Software Process Improvement Plan (SPIP) must be formally established throughout the project.

PRACTICE ESSENTIALS:

- Writing the process improvement action plan
- Acting on the process improvement action plan
- Assessing results

PROBLEM ADDRESSED:

Action plans are of no use if they're only developed and not effectively carried out.

REQUIREMENTS:

- A high-visibility kickoff
- Empowered software team leaders
- Rewards for process improvement successes
- Organizational awareness and conformance

STATUS CHECKS:

- Are concerned project members aware of the software process and its progress?
- Are views of product progress available and current?

The SPIP is only as good as its encouragement and enforcement.

PRACTICE: CLOSING THE LOOP FOR SOFTWARE PROCESS IMPROVEMENT

Process improvement requires continuous iterative feedback to all vested parties.

PRACTICE ESSENTIALS:

- Measurements throughout the process improvement effort, including system cycle time, reassessment results, and training resources spent as planned
- Feedback on process improvement progress available to all participants in the software improvement effort, including sponsors, software engineering process group members, process engineers, support and steering committee members, senior software technologists, and all practitioners

PROBLEM ADDRESSED:

In the hectic environment of a software development project, good practices can get lost in races to impractical, sometimes impossible, deadlines.

REQUIREMENTS:

- Maintaining a current process asset library

STATUS CHECKS:

- What is the status of the highest-priority Process Improvement Plan?

Like all communication, feedback is best when clear and concise.

7. SOLICITATION AND CONTRACTING BEST PRACTICES

Solicitation and contracting goals include:

- Delivering a system that works well in large-scale software developments
- Aligning government employees' and taxpayers' demands for value
- Eliminating wasteful paperwork and outdated procedures
- Using criteria-based incentives

Solicitation and Contracting Best Practices include:

- Management of COTS, Reuse, and Emerging Technologies
- Employing a Customer/Contractor Integrated Product Team (IPT)
- Use of Periodic Demos
- Utilizing Software Development Capability Evaluation (SDCE)

PRACTICE: MANAGEMENT OF COTS, REUSE, AND EMERGING TECHNOLOGIES

The government/contractor team should agree to a process that rewards competence and allows risk to be managed when Commercial Off-the-Shelf (COTS) software, reuse, and other emerging technologies are to be used on an impending procurement.

PRACTICE ESSENTIALS:

- Rewards for achieving emerging-technology goals expressed in fee-sharing formulas accepted by a joint customer/contractor team
- Federal Acquisition Regulation (FAR) clauses that enable use of emerging technologies
- Risk mitigation approaches that are compatible with:
  - User requirements
  - Contractor competition
  - Negotiated rights of ownership and their criteria

PROBLEM ADDRESSED:

The overall goals of COTS, reuse, and use of other emerging technologies are to reduce cost and/or improve quality.

REQUIREMENTS:

- Focus for COTS, reuse, and emerging technologies on:
  - Cost avoidance
  - Schedule reduction
  - Quality improvement
- Solicitation clauses and contract terms and conditions with specifications concerning emerging technologies

STATUS CHECKS:

- Has the overall project cost been reduced (although initial cost to develop reusable assets is higher)?
- Has time to market been reduced because there are fewer new assets to build?
- Is quality higher because reusable assets and COTS are more stable with fewer errors?

PRACTICE: EMPLOY A CUSTOMER/CONTRACTOR INTEGRATED PRODUCT TEAM

A customer/contractor IPT should be established and sustained to improve communication on all projects, to define the risk issues, and to define responsibilities relevant to the identification and

mitigation of risks that affect cost and schedule growth.

PRACTICE ESSENTIALS:

- Members who are multifunctional and experienced
- Alternates to stand in for members who cannot attend meetings
- Preferred mechanisms of communication identified and established within the IPT to the rest of the project organization, and from the rest of the project organization to the IPT
- A defined charter with goals and responsibilities clearly stated
- A team for each key program area or risk

PROBLEM ADDRESSED:

Members of different organizations must compete for resources, which can generate conflict.

REQUIREMENTS:

- A risk management team that begins pre-contract award and is maintained throughout the project
- Prioritized risks associated with equipment, software, and facilities

STATUS CHECKS:

- Are viewpoints other than the project team's involved in the risk assessment process?
- Is the IPT able to conduct an assessment of progress and risk without additional data, analysis, cost, or extension of the schedule?

IPT members must be straightforward in communication of intentions before, during, and after IPT meetings, and with all functional areas—no hidden agendas.

PRACTICE: USE OF PERIODIC DEMOS

Periodic demos can reduce risk, especially during solicitation and procurement phases of a contract.

PRACTICE ESSENTIALS:

- Demonstration of unrealized product functions, operations, and interfaces

PROBLEM ADDRESSED:

Proposed ideas are more easily evaluated when they are visible.

REQUIREMENTS:

- To make proposed functions visible
- To show contractor's ability to use proposed technologies prior to contract award

STATUS CHECKS:

- Has the proposed technology been successfully demonstrated in a representative operational environment?
- Has the proposed technology been successfully integrated in the evolving program?

- Have product functions been evaluated in a critical item test?

During software development, risk items identified in the Request for Proposal (RFP) should be mitigated through critical item demo testing.

PRACTICE: UTILIZE SOFTWARE DEVELOPMENT CAPABILITY EVALUATION

The Software Development Capability Evaluation (SDCE) methodology provides a structured approach for assessing an organization's capability to develop software for mission-critical computer resources.

PRACTICE ESSENTIALS:

- Methodology for soliciting a contractor qualified to develop software in accordance with requirements
- Evaluation of contractor's technical and management process to ensure consistent execution at the highest level possible

PROBLEM ADDRESSED:

Contracts are awarded to bidders based solely on proposals that, in many cases, do not provide insight into the contractor's actual capability to develop the specific software needed for the project.

REQUIREMENTS:

- Professional team with in-depth experience in software acquisition
- Professional team with experience in key project domain areas

STATUS CHECKS:

- Is there a commitment to use of SDCE?
- Is an SDCE team in place?
- Has the SDCE been tailored to the specific project?

As defined in Air Force Materiel Command Pamphlet (AFMCP) 663-103, 15 June 1994, SDCE:

- *Was developed with industry as a partner*
- *Is independent of military and industry development and management standards*
- *Focuses on the specific needs of the specific acquisition program; for example, it must be tailored to meet specific program needs*

Best Practices by Project Management Area	Principal Best Practices									
	Formal Risk Management	Agreement on Interfaces	Formal Inspections	Metrics-based Scheduling and Management	Binary Quality Gates at the Inch-Pebble Level	Program-wide Visibility of Progress vs. Plan	Defect Tracking Against Quality Targets	Configuration Management	People-Aware Management Accountability	Enabling Practice/Mechanism
Risk Management										
Establish Management Reserves for Risk Resolution	X									
Implement Metrics-based Risk Decisions	X									
Perform Continuous Risk Management	X									
Formalize Risk Tracking and Review	X									
Manage Impact of External Dependencies	X	X								
Planning										
Quantitative Software Estimation/Verification				X						X
Joint Team Involvement		X								
Activity Planning										X
Data Requirements		X								
Program Visibility										
Practical Project-Oriented Software Measurement Process				X						
Issue-Driven Measures				X		X				
Internal Engineering Analysis Process						X				
Effective Communication Structure			X			X				
Program Control										
Test Methodology					X		X			X
Regression Testing					X		X			X
Computer-Aided Software Testing					X		X			X
Error Source Location							X			
Independent Verification and Validation (IV&V)			X				X			X
Quality Gate Completion Criteria					X					
Configuration Management Coverage								X		
Requirements Change Management								X		
Baseline Methodology								X		
Technical Quality Assurance					X					

Figure 6.1 Relationship between Principal Best Practices and Best Practices by Project Management Area

Engineering Practices and Culture										
Include User in a Multi-Disciplined Requirements Support Team		X								
Encourage Compatible Analyses and Design Methods										X
Encourage Software Architecture Definition and Maintenance		X								
Encourage Requirements Engineering Process that Includes Use of Prototypes, Models and Simulations	X									
Encourage Proactive Change Impact Analysis								X		
Plan for Domain Engineering in Acquisitions		X								X
Encourage Use of Clean Room Techniques										X
Enterprise Practices Tailored to Projects										X
Encourage Use of Software Development Standards such as MIL-STD-498			X						X	
Assessing Organizational Effectiveness									X	
Process Improvement										
Identifying and Fostering Sponsorship									X	X
Establishing/Maintaining the Framework for Progress Improvement										X
Assessing/Reassessing an Organization's Process Quality										X
Developing a Software Process Improvement Plan										X
Institutionalizing the Software Process Improvement Plan										X
Closing the Loop for Software Process Improvement										X
Solicitation and Contracting										
Management of COTS, Reuse and Emerging Technologies										X
Employ a Customer/Contractor Integrated Project Team	X									
Use of Periodic Demos		X								X
Utilize Software Development Capability Evaluation (SDCE)										X

- *Relies on and encourages use of offeror's internal software development processes*
- *Requires evidence of past performance*
- *Requires no Request for Proposal levels for source selection*
- *Solicits and supports contractual commitment to the offeror's proposed software development processes*

In the event that a program manager desires additional information, the Network will provide both source materials and access to experts. We would greatly appreciate receiving your comments and suggestions (preferably by e-mail).

E-MAIL: BEST@SPMN.COM PHONE: (703) 521-5231 FAX: (703) 521-2603

Best Practices	Control Panel Gauges														
	Cumulative Earned Value Delivered	Cumulative Months	Cumulative Dollars	CPI	TCPI	"Abba Chart"	Tasks	Quality Gates Progress	Aggregate Requirements Growth	Voluntary Turnover	Overtime Hours	Risk Impact	Risk Liability	Warnings	Defects by Activity
General Application-No Gauge Impact															
Establish Management Reserves for Risk Resolution												X	X		
Implement Metrics-based Risk Decisions												X	X		
Perform Continuous Risk Management												X	X		
Formalize Risk Tracking and Review												X	X		
Manage Impact of External Dependencies								X				X	X		
Planning															
Quantitative Software Estimation/Verification	X	X	X	X	X	X		X							
Joint Team Involvement									X						
Activity Planning		X	X				X								
Data Requirements									X						
Program Visibility															
Practical Project-Oriented Software Measurement Process	X	X								X	X				
Issue-Driven Measures				X	X	X	X			X	X				X
Internal Engineering Analysis Process							X								
Effective Communication Structure										X	X			X	
Program Control															
Test Methodology															X
Regression Testing															X
Computer-Aided Software Testing															X
Error Source Location															X
Independent Verification and Validation (IV&V)															X
Quality Gate Completion Criteria	X			X	X	X	X					X			
Configuration Management Coverage															X
Requirements Change Management															X
Baseline Methodology															X
Technical Quality Assurance															X
Engineering Practices and Culture															
Industry User-in-a-Multi-Disciplined Requirements Support Team									X						

Figure 6.2 Relationship between Control Panel Gauges and Best Practices

Encourage Compatible Analyses and Design Methods									X							
Encourage Software Architecture Definition and Maintenance									X							
Encourage Requirements Engineering Process that Includes Use of Prototypes, Models and Simulations									X							
Encourage Proactive Change Impact Analysis																X
Plan for Domain Engineering in Acquisitions																X
Encourage Use of Clean Room Techniques																X
Enterprise Practices Tailored to Projects																X
Encourage Use of Software Development Standards such as MIL-STD-498															X	X
Assessing Organizational Effectiveness																X
Process Improvement																
Identifying and Fostering Sponsorship																X
Establishing/Maintaining the Framework for Progress Improvement																X
Assessing/Reassessing an Organization's Process Quality																X
Developing a Software Process Improvement Plan																X
Institutionalizing the Software Process Improvement Plan																X
Closing the Loop for Software Process Improvement																X
Solicitation and Contracting																
Management of COTS, Reuse and Emerging Technologies																X
Employ a Customer/Contractor Integrated Project Team																X
Use of Periodic Demos																X
Utilize Software Development Capability Evaluation (SDCE)																X

CHAPTER 7

PROJECT CAVEATS

PROJECT CAVEATS

The following software management caveats are lessons learned from software and hardware/software projects gone awry.

1. Don't expect schedule compression of 10 percent or more compared with the statistical norm for similar projects.

$$\text{Schedule Compression Percentage} = \left\{ 1.00 - \left[\frac{\text{Calendar Time Scheduled}}{\text{Nominal Expected Time}} \right] \right\} \times 100$$

Nominal Expected Time is a function of total effort expressed in person months.

2. Don't justify new technology by the need for schedule compression.

New technology is any tool or development method not used before by the staff and management of the current project.

3. Don't force customer-specific implementation solutions on the program.

Implementation technology includes internal design, hardware/software partitioning, reuse plans, etc.

4. Don't advocate use of silver bullet approaches.

A new approach qualifies as a silver bullet (as described by Frederick Brooks in his classic essay, "No Silver Bullet: Essence & Accidence of Software Engineering") if it claims to have a 20 percent or greater effect on productivity and is as yet untried by program staff on projects of the same size and scope as the one they are about to undertake.

5. Don't miss an opportunity to move items that are under external control off the critical path.

When possible, schedule external dependencies that are on the critical path so that their impact on the project is lessened.

6. Don't bury all project complexity in software as opposed to hardware.

During the early stages of project work, it is tempting to allocate the portions of the requirement that are not yet fully understood to software, which assures that software will become the critical path problem as the project moves toward completion. No function that is incompletely specified should be allocated to the software. Hardware/software partitioning cannot be done effectively unless the specification is reasonably complete.

7. Don't conduct critical system engineering tasks without sufficient software engineering expertise.

Hardware/software tradeoffs cannot be effectively completed with expertise from only one of the two areas.

8. Don't expect to achieve an accurate view of project health from a formal review attended by a large number of unprepared, active reviewers.

The review process suffers from diminishing returns when the number of participants goes much beyond a dozen. All reviewers need to be prepared in detail to deal with all, or at least key, parts of the product under review Extensive preparation for active participation in the review should be the entry price for attendance.

9. Don't expect to recover from a schedule slip of 10 percent or more without a 10 percent or greater reduction in software functionality to be delivered.

Software functionality can be quantified in terms of Function Points or another specifications-based metric of system size. Recovery from significant schedule slip should be expected only when accompanied by comparable reduction of system scope resulting in reduced Function Points.

GLOSSARY

GLOSSARY

Abba chart • A graph (named for Wayne Abba) that is composed of four different indicators showing trends in historic and projected efficiency to date. Also known as the Total Program Performance Chart.

Acceptance criteria • The list of requirements that a program or system must demonstrably meet before customers accept delivery. Late changes in acceptance criteria or hidden criteria derived from explicit criteria cause problems for software projects.

Acceptance test • A form of testing in which users exercise software prior to accepting it for production runs. The IEEE definition assumes that acceptance testing will be formal.

Actual cost • The cumulative actual cost incurred on the project to date.

ACWP • Actual Cost of Work Performed.

Alpha tests • The first tests of a product, using real input, when it is still in an unfinished state. Alpha tests are usually internal to an organization and are followed by beta tests.

Architecture • The structure and interrelation of a system's components, including the relation of the interface to its operational environment.

Audit • An independent review of product development and process execution to confirm that they conform to standards, guidelines, specifications, and procedures.

BAC • Budget at Completion.

Baseline • A specification or product that has been reviewed and agreed on, and that thereafter serves as the basis for further development. A baseline can be changed only through change control procedures.

BCWP • Budgeted Cost of Work Performed *(see* Earned Value*)*.

BCWS • Budgeted Cost of Work Scheduled.

Beta tests • Testing a product in its intended environment with the results used for their intended application.

Binary acceptance criteria • A list of requirements that a deliverable must completely satisfy before moving on to the next activity or task.

Budget at Completion (BAC) • The total original budget for a project, which is the maximum value on the Control Panel Earned Value gauge.

Budgeted Cost of Work Scheduled (BCWS) • The cumulative planned value, which is the total value of work that was originally scheduled for completion by the end of a reporting period.

CASE (Computer-Aided Software Engineering) • The industrialization of software engineering techniques and computer technology to improve and automate the practice of information systems development.

Clean Room • A process that uses formal program specification and verification and statistical software quality assurance to create high-quality software.

Constructive Cost Model (COCOMO) • A closely related family of software cost estimating models developed by Dr. Barry Boehm of TRW.

Code complexity • The complexity of software code, usually affected by factors such as cohesion, coupling, modularity, and module complexity factors including SLOC, nested loops, global variables, and GOTO statements.

Complexity estimate • A numerical prediction of the probable number of interrelated factors that cause projects to be viewed as complex. Models that measure logic, code, and data complexity include the McCabe cyclomatic and essential complexity metrics, the NPATH complexity metric, SPQR, and CHECKPOINT.

Component • The collection of programs and modules that perform a single, identified technical or business function. Examples of components include the scheduler of an operating system or the parser of a compiler.

Configuration Management (CM) • The process of identifying and defining the deliverable product set in a system, controlling the release and change of these items throughout the system life cycle, recording and reporting the status of product items and change requests, and verifying the completeness and correctness of the product items.

Contingency factor • A reserve amount that companies add to cost estimates and budgets to cover unanticipated expenses and that acts as a buffer against estimating errors. Normal contingency factors would be 35 percent added to cost estimates produced during requirements, 25 percent if produced during design, 15 percent if produced during coding, and 5 percent if produced during testing.

Cost factors • Parameters that influence the amount of resources needed to accomplish a job.

Cost overrun • Situation where the actual cost exceeds the estimated or budgeted amounts on projects or deliverables.

Cost Performance Index (CPI) • The Control Panel gauge that shows how efficiently the project team has turned costs into progress to date.

COTS • Commercial Off-the-Shelf (often used in reference to software).

Critical path • The set of activities that must be completed in sequence and on time if the entire project is to be completed on time.

Cumulative defect removal efficiency • The percentage of software defects found by all reviews, inspections, and tests prior to software delivery compared to all defects found during development and by users in a fixed time interval, such as the first year of operation.

GLOSSARY

Cyclomatic complexity • An aspect of the McCabe complexity metric that looks at the control flowgraph of a program and determines software complexity based on the minimum number of paths.

Defect • A problem or "bug" that, if not removed, could cause a program to either produce erroneous results or otherwise fail.

Defect potential • The probable number of defects from all causes that will be encountered during the development and production of a program or system. Defect potential is enumerated as the sum of five defect categories: requirements, design, coding, documentation, and bad fixes or secondary defects.

Defect prevention • Technologies and techniques that minimize the risk of human error. Defect prevention techniques include structured analysis and design, high-level languages, participation in Joint Application Design sessions, and reviews and inspections.

Defect removal • Activities that are aimed at removing defects from software, including walkthroughs, reviews, inspections, editing, and all forms of testing. For military projects, defect removal is the second most expensive activity, with paperwork being the most expensive. A synergistic combination of defect prevention and defect removal can yield dramatic improvements in the quality of delivered software.

Defect removal efficiency • The number of defects removed by a specific operation, such as a code inspection, review, or test phase, compared to the total number of defects found during software development and the first year of operation.

Defect severity • Classification of defects into categories such as critical, serious, moderate, cosmetic, or tolerable. Classification may also be numeric, ranging from 1 (high severity) to 4 or 5.

Deliverable • A tangible, physical object that is the output of a software development task. Examples of deliverables include requirements documents, specifications, test cases, and source code. There are also synthetic deliverables such as Function Points or Feature Points.

Design • The tasks associated with specifying and sketching out the features and functions of a new application prior to formal coding.

EAC • Estimate at Completion.

Earned Value (EV) • A means of evaluating budgetary performance by relating actual expenditures to technical achievement as measured by a milestone accomplishment scheme. EV may be used interchangeably with BCWP.

Effort • The person-months or person-years of work by all job classifications on the software product (design, coding, inspection, testing, documentation, and supervision).

Embedded software • Software for an embedded system. An embedded system is integral to a larger system whose primary purpose is not computational; for example, a computer system in an aircraft or a rapid transit system.

Error source location • The backward exploration of the cause of an error or defect from point of occurrence to ultimate reason.

Estimate at Completion (EAC) • The maximum value on the Control Panel Actual Cost gauge, which represents the current best estimate for total cost of the project.

Function Point (FP) • A unit of measure of software size based on owner and user requirements stated in the requirements specification.

Gantt chart • A chart (named for Henry Laurence Gantt) that consists of a table of project task information and a bar chart that graphically displays project schedule, depicting progress in relation to time and often used in planning and tracking a project.

Inch-pebble • The lowest level of an activity network consisting of a defined pass/fail task that can be accomplished within a short period of time, such as two weeks.

Independent Verification and Validation (IV&V) • Verification and validation of a software product by a group other than the one that created or implemented the original design.

Inspections • Visual examinations to detect errors and standards violations in requirements, design, code, user documentation, test plans and cases, and other software development products.

IPTs • Integrated Product Teams.

Interface • The boundary between two programs, two pieces of hardware, or a computer and its user.

Joint Application Design (JAD) • A defect-prevention technique determining requirements for a software project through structural, joint sessions of users and developers.

Kiviat graph • A multifaceted graphic representation technique used to display the results of many changing variables simultaneously. Kiviat graphs are used to display productivity, quality, and other targets together. The graph appears as a star-like set of lines radiating from a central point. This central point provides the zero point or origin of the lines, each of which represents a norm for a particular metric.

Main software build • A stage in the project life cycle, following functional design, that begins with detailed program design and continues through coding and system testing until the system is operational.

Manpower buildup • The rate of building up personnel on a project. This should match the rate at which the project leaders identify problems and assign them to the staff.

Maximum development time • The limit beyond which it is unlikely that a project can be successfully completed.

GLOSSARY

Metrics • Means by which software engineers measure and predict aspects of processes, resources, and products that are relevant to the software engineering activity.

Minimum development time • The limit below which it is impractical to attempt to develop a system given its size, level of productivity, and rate of manpower buildup.

Peer review • A type of review that is conducted by peers to evaluate a product, such as a segment of design or unit of code. Peer reviews may be formal or informal. Walkthroughs and inspections are often conducted as peer reviews.

PERT chart • A chart (resembling a flow chart) in which a box represents each project task, and a line connecting two boxes represents the relationship between two tasks.

Process productivity measure • A measure, obtained from past projects, of the effectiveness of an entire project or organization in developing software.

Product functionality • The number of Source Lines of Code created. This value is calculated using models for estimating size.

Productivity • A measure of the amount of Source Lines of Code that can be delivered per person-month.

Program documentation • All on-line and hard-copy information supporting the system's contractual agreement, design, build, operation, and maintenance.

Prototyping • A process in which partial versions of a program are created to aid in designing the final product.

Quality • The totality of features and characteristics of a product that bear on its ability to satisfy given needs.

Quality assurance • All the planned and systematic actions necessary to provide adequate confidence that a product or service will satisfy given requirements for quality.

Quality gate • A predefined completion criterion for a task including audits, walkthroughs, and inspections, that provides an assessment of progress, processes used, and project products.

Rayleigh curve • A roughly bell-shaped curve that represents the buildup and decline of staff power, effort, or cost, followed by a long tail representing staff power, effort, or cost devoted to enhancement or maintenance.

Regression testing • Selective retesting to detect faults introduced during modification of a system.

Requirements growth • The increase between baselined and current documented requirements.

Reuse • The ability to make additional use of standard parts or components such as reusable code, design, architectures, and test cases.

Review • An examination (formal or informal) of the specification, code, or another deliverable from a software project.

Risk • The probability that a software project will experience potential hazards that will affect the schedule or completion of the project.

Risk reserve • Money and time held in reserve to be used in the event that risks occur.

SAC • Schedule at Completion.

Silver bullet • A single tool or method expected to significantly improve software productivity.

Size • Delivered, executable SLOCs. Comment statements or blank lines are excluded from the size.

Slip • The amount of time that a deliverable or product is late from its originally scheduled date.

Source Line of Code (SLOC) • A physical line (non-comment, non-blank) of deliverable source statements.

Stakeholders • People, organizations, and existing systems that are affected by, or that influence, the proposed system development or enhancement, including the customer and users.

To-Complete Performance Index (TCPI) • The Control Panel gauge that shows the future projection of the average productivity needed to complete the project within an estimated budget.

Total quality management • A method of removing waste by involving everyone in improving the way things are done. Total quality management techniques can be applied throughout the company, and are equally useful in all departments whether production- or service-oriented.

Voluntary staff turnover • A measurement of employees the project wants to keep, but who have chosen to leave.

Walkthrough • A review process in which a designer or programmer leads one or more members of the development team through a segment of design or code that he or she has written, while the other members ask questions and make comments about technique, style, possible errors, violation of development standards, and other problems.

Work Breakdown Structure (WBS) • The product- or activity-oriented hierarchy tree depicting the elements of work that need to be accomplished in order to deliver an end product to the customer.

APPENDICES

OFFICE OF THE SECRETARY OF DEFENSE

WASHINGTON, D.C. 20301

JUL 9 1994

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
ATTN: SERVICE ACQUISITION EXECUTIVES
VICE CHAIRMAN, JOINT CHIEFS OF STAFF

SUBJECT: Software Acquisition Best Practices Initiative

Current DoD software acquisition practices have not proven to provide an effective framework for managing the acquisition of large-scale software development and maintenance programs that are an essential part of our increasingly complex weapons systems. Although many excellent practices for effectively managing such programs exist in both industry and government, their understanding and use within our software acquisition programs is not widespread. The April 5, 1994, memorandum from the Director, Acquisition Program Integration, OUSD(A&T) directed Service Acquisition Executives to the importance of integrating these practices into their software acquisition processes.

We share these concerns, and together have established the *Software Acquisition Best Practices Initiative* to improve and restructure our software acquisition management process. The purpose of this initiative is to:

- Focus the Defense acquisition community on employing effective, high-leverage software acquisition management practices;
- Enable Program Managers to focus their software management efforts on producing quality software, rather than on activities directed towards satisfying regulations that have grown excessively complex over time;
- Enable Program Managers to exercise flexibility in implementing Best Practices within disparate organizational program cultures, and,
- Provide Program Managers and staff with the training and tools necessary to effectively use and achieve the benefits of these practices.

This initiative will identify practices used by successful software projects from both government and industry, and will expand and support the efforts now underway by the Software Program Managers Network to identify and convey these practices. These practices are to be defined as criteria-based practices. The Defense Acquisition Management process, managed under DoD Instructions 5000.2, and 8060, and the associated program review processes will be modified to effectively implement them. Successfully accomplishing this initiative is fundamental to achieving urgently needed improvements to our software acquisition practices.

The Director, Test and Evaluation, OUSD(A&T) and the Deputy Assistant Secretary (C3I Acquisition) are directed to jointly define, implement, and manage this initiative. Please designate a representative having substantial experience in directly managing large-scale Mission-Critical or C3I software development projects to participate in formulating and implementing this initiative. Provide your designee's name, phone and resume by July 29, 1994, to the Director, Test and Evaluation, OUSD (A&T). Please direct questions to the Network's Coordinator, Norm Brown, at (703) 523-5231 (e-mail: nbrown@nadc.navy.mil).

Noel Longuemare
Under Secretary of Defense
(Acquisition and Technology) (Acting)

Emmett Paige, Jr.
Assistant Secretary of Defense
(Command Control, Communications, and Intelligence)

OFFICE OF THE UNDER SECRETARY OF DEFENSE

3000 DEFENSE PENTAGON
WASHINGTON DC 20301-3000

6 APR 1998

MEMORANDUM FOR SERVICE ACQUISITION EXECUTIVES
PROGRAM EXECUTIVE OFFICERS
PROGRAM MANAGERS

SUBJECT: Software Acquisition Management "Best Practices"

Software has clearly become a major cost, schedule, and performance driver of virtually all of our weapons, command & control, and information systems. Many of these programs can benefit through effective use of "Best Practices" successfully being used in DOD and industry large-scale software intensive programs. Widespread understanding and appropriate effective use of such practices is essential to achieving the necessary sustained improvement in software acquisition management.

Service Acquisition Executives and Program Executive Officials can contribute greatly by adopting measures which emphasize utilization of such best practices in the acquisition management process. As something that may help in this regard, I've attached a checklist of some of the essential best practices which are generally appropriate to effective acquisition management of large-scale, software intensive systems.

The Software Program Managers Network (described in the attached ASD(C3I) memorandum of July 16, 1993) is now underway identifying these best practices and related lessons learned and conveying them to program managers and staff. I join with Assistant Secretary Paige and others in encouraging your active participation (membership form attached), as well as that of your industry counterparts, in the Network's activities. Your successes can benefit others, and perhaps theirs can benefit you. The Network needs your help and involvement in collecting and conveying these successes and lessons learned.

Gene H. Porter
Director, Acquisition
Program Integration

attachments

PRINCIPAL DEPUTY UNDER SECRETARY OF DEFENSE

3010 DEFENSE PENTAGON
WASHINGTON, DC 20301-3010

ACQUISITION AND TECHNOLOGY

JUN 5 1996

MEMORANDUM FOR COMPONENT ACQUISITION EXECUTIVES

SUBJECT: Improved Cost and Schedule Performance Management

Effective program cost and schedule management depends on establishment by contractors of reliable performance measurement baselines. The baseline must capture the entire technical scope of work consistent with contract schedule requirements, and have adequate resources assigned. An accepted recommendation in DoD Inspector General Audit Report No. 91-067, "Use of Cost and Schedule System Data," addressed the need for better baseline evaluation.

For contracts requiring compliance with the Department of Defense Cost/Schedule Control Systems Criteria (C/SCSC), I expect program managers and their technical staffs to review the baselines established by their contractors within six months after contract award to assure that planning and budgeting are accurate at the cost account level. Please provide a copy of your implementing guidance within thirty days to the Director, Acquisition Program Integration.

These integrated baseline reviews are not a substitute for normal C/SCSC reviews, and will be conducted in coordination with DoD Component C/SCSC specialists. Two objectives of the baseline reviews are to reduce the number of C/SCSC reviews required, and to improve use of cost performance data by contractor and government managers. Similar but less comprehensive reviews will be performed for contracts subject to Cost/Schedule Status Report requirements.

A good example of the integrated baseline review concept was implemented by the Army Program Executive Office, Missile Defense, in 1993. A report on the process is attached. While other approaches may differ, their success requires the level of executive leadership and thorough preparation apparent in the Army report.

[signature]

R. Noel Longuemare

Attachment

FOUNDATION FOR SOFTWARE CONTRACTING

Development and Maintenance of Large-Scale Software

PURPOSE

- To reverse a chronic problem which drives very large cost overruns, schedule slips, and delivers systems which work poorly at best in large-scale software developments.
- To unleash the full potential of software technology that up to now has been limited by management problems.
- To eliminate highly wasteful practices and produce real value for taxpayers.
- To align goals of industry and government; to eliminate useless regulations; to emplace successful software industry practices in the defense software industry.

METHOD

- Drive defense software contractors to utilize known best practices, eliminating wasteful activity, optimizing operations, and leading to productivity gains and substantial overall cost savings. Some benefits include:
  - Reduced documentation
  - Reduced rework
  - Streamlined management
  - A more reliable product at a lower cost to the taxpayer.
- Revise the contracting payment method to produce overall net cost savings by using criteria-based contract incentive payments.
- Make DoD software acquisition more competitive by facilitating commercial industry participation.
- Eliminate the burden of specialized government operations and allow company management to concentrate on delivering a high-quality product.

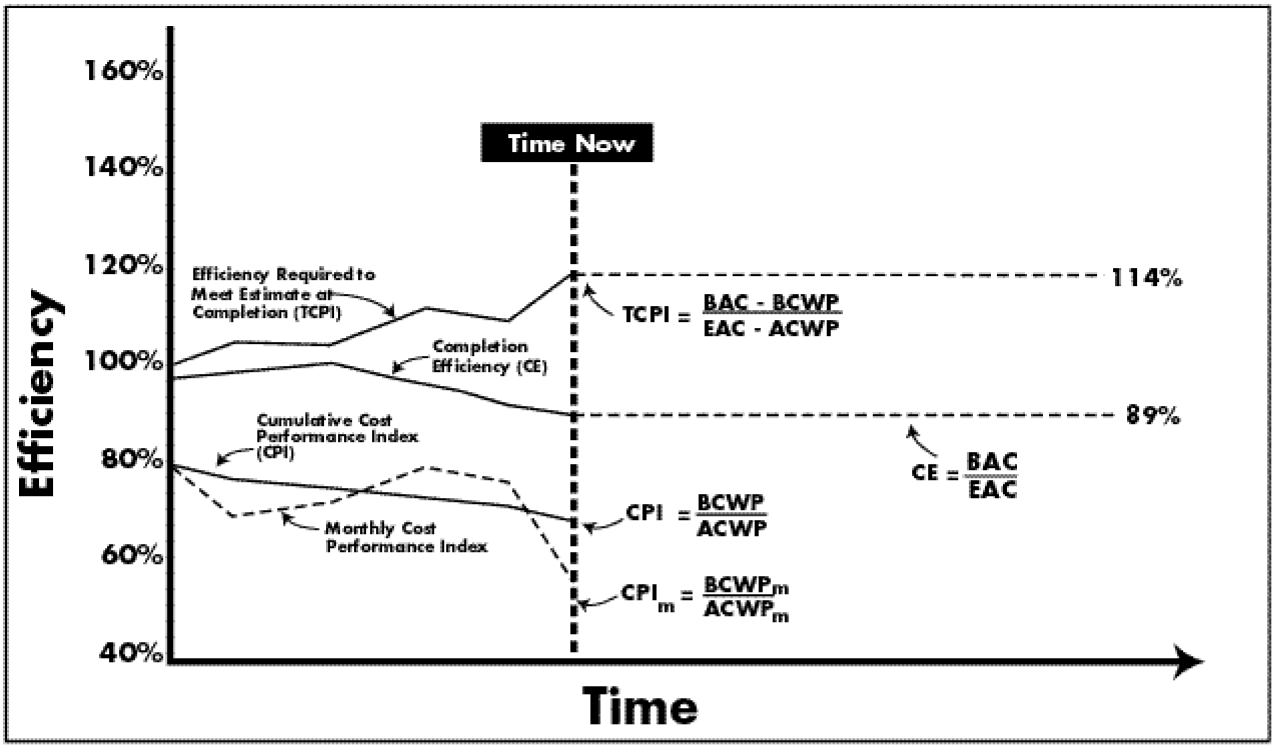
RESULT

- Because total defense costs decrease, and net profits for highly effective companies increase, this means good news for the best defense software producers, and strong companies for defense.
- Savings over five years, conservatively estimated, range from 10% - 20% of DoD annual software cost, which was approximately $42 billion in 1994.

APPENDIX C

TOTAL PROGRAM PERFORMANCE EFFICIENCY

"Abba Chart"



BEST PRACTICES INITIATIVE CONTRIBUTORS

Participant Organizations

Ada Pros, Inc.
Aerospace Corporation
Aerospace Industries Association
AIL Systems
Apple Computer
Arizona State University
ASC
Atlantic Systems Guild, Inc.
Auburn University
BDM
Bell Atlantic
Boeing
Borland
CADRE Technologies, Inc.
Ceridian Corporation
Computers & Concepts Associates
Computer Sciences Corporation (CSC)
Coopers & Lybrand
David Maibor Associates
Digicomp Research Corporation
Digital Equipment
Dupont
EDS
Electronic Industries Association
FAA
General Electric
GTE
Harris Corporation
Honeywell, Inc.
Hughes
IBM
ITT
Keane Federal Systems
Kodak
Lockheed Martin Corporation
Logicon
Loral
Martin Marietta
McCabe & Associates
McDonnell Douglas

Participant Organizations

MCI

Mitre

Mobil Oil

Motorola

NASA

NCR

National Security Industrial Association

Oshkosh Truck Corporation

Pitney Bowes

Predictive Technologies

QSM, Inc.

Rational

Raytheon

Rockwell

SAIC

SEI

Software Productivity Research, Inc.

Software Productivity Solutions, Inc.

Sprint

State Farm Insurance

Sun

Sverdrup Corporation Space

Technologies Applications, Inc.

Tecollote Research, Inc.

Texas Instruments

The Analytic Sciences Corporation

Tracor

Union Pacific Technology

Unisys

United Technologies

University of California at Berkeley

University of Maryland

U.S. Air Force

U.S. Army

U.S. Coast Guard

U.S. Marine Corps

U.S. Navy

Vanguard Research

Westinghouse

BEST PRACTICES INITIATIVE CONTRIBUTORS

ISSUE PANELS

Risk Management

- **Brian Koster** (NAVAIR)
- **Anna Deeds** (NAVSEA/PEOTAD)
- **Frank Sitsi** (SEI)
- **Jerry Lecroy** (MITRE)
- **Stan Levine** (Army/PM CH5)
- **Tom Conrad** (Navy-NUWC)
- **Bob Hegland** (Army)
- **Frank Gregory** (Army-MLRS Proj. Office)
- **Dean Eliott** (Navy/China Lake)
- **John Hoyem** (Navy/China Lake)
- **Jim Huskins** (Air Force)
- **LTC Carlos Galvan** (Air Force)
- **George Prosnic** (DSMC)
- **LCDR Mike Borowski** (COTF)
- **Steve McComas** (NAWC)
- **Dario DeAngelis** (Logicon)
- **Jay Bach** (Boreland)
- **Sue Markel** (TRW)
- **Tony Hutchings** (Digital)
- **Tom Duggan** (Mobil Oil)
- **John Travalent** (Unisys)
- **David Hendrickson** (Honeywell)
- **Raymond Curts** (SWL)

Planning & Baselining

- **Carl Hall** (Navy/China Lake)
- **CDR Howard Taylor** (NSA)
- **Phil Acuff** (AMSMI/RD/MG/CT)
- **Norma Stopyra** (JLC)
- **Austin Huangfu** (DoD) (OT&E)
- **Sherwin Jacobson** (DSMC)
- **Debra Martin** (Navy-SPAWAR)
- **CAPT Gregor** (C41)
- **Perry DeWeese** (Lockheed Martin Aeronautical Systems)
- **Patti Shishido** (TRW)
- **Greg Farham** (Lockheed Martin)
- **Ray Zachary** (Loral)
- **Oleh Kostetsky** (Predictive Tech)
- **Connie Palmer** (McDonnell Douglas)
- **Teri Snyder** (Hughes)
- **Wesley Shellenbarger** (Unisys)
- **David Swope** (Sanders)
- **Richard Law** (Lockheed Martin)
- **Richard Fanning** (Hughes)

Program Visibility

- **Jack McGarry** (Navy/NUWC)
- **COLLarry Sweeney** (HQ/AFMC)
- **Ken Kelley** (DISA)
- **Bill Agresti** (Mitre)
- **Betsy Bailey** (IDA)
- **Ed Primm** (NSWC/PH/ECO)
- **Harpal Dhama** (Mitre-Bedford)
- **Andrew Chruscicki** (RL/C3-CB)
- **Norm Schneldwind** (Navy-PG School)
- **Anita Carlton** (SEI)
- **Jim Blackwelder** (Navy-NSWC Dahlgren)
- **Jim Bischoff**
- **Tony Guido** (NASC)
- **Paul Reindollar** (Lockheed Martin)
- **David Card** (Sps, INC.)
- **Frank McGarry** (CSC)
- **Kyle Rone** (Loral)
- **Joseph Dean** (Tecollote Research Inc.)
- **Deborah DeToma** (GTE Government Systems Corp.)
- **Peter Dyson** (Software Productivity Solutions, Inc.)
- **Edward F. Weller** (Motorola)
- **John T. Harding** (Software Technology Transition)
- **Bob Sulgrove** (NCR-Dayton)
- **Wendy Shutter** (Digicomp Research Corp)
- **Bob Rova** (Hughes)
- **Rick Cooperman** (Hughes)

Program Control

- **Ray Paul** (OUSD (A&T) T&E
- **Cindy King** (FAA)
- **Larry Baker** (DSMC)
- **Ron Green** (NASA–Huntsville)
- **CAPT Bruce Freund, USN** (NAVSEA)
- **Bill Brykczynski** (IDA)
- **Dr. Luqi** (Navy PG School)
- **George Hurlburt** (NAWC)
- **Margaret Powell** (ASN [RDA]/NISMC)
- **Luke Campbell** (Navy-NATC)
- **George Axiotis** (NAVSEA)
- **CAPT Richard Poligala** (AFOTEC)
- **MAJ Thornton** (MCOTEA)
- **William Farr** (NSWC)
- **Jeanne LeFevre** (Unisys)
- **Leonard Tripp** (Boeing Commercial)
- **Kathy James** (CTA)
- **Boris Beizer** (Author)
- **Peter Kind** (SEI)
- **Tony Schumskas** (BDM)
- **Joyce Jakaltis** (ASC)
- **Fred Hall** (IEI)
- **C.V. Ramamocrthy** (UCB)
- **Danny Shoup** (Boeing)
- **Pratap Chillakanti** (Hewlett-Packard)

ISSUE PANELS

Engineering Practices & Culture

- **Rubin Pitts** (Navy-NSWC/Dahlgren)
- **Tara Regan** (USA/SSDC)
- **George Robinson** (Navy)
- **Mary Lou Urban** (Mitre)
- **Mike Rice** (NAVSEA)
- **J. Alberto Yepez** (Apple)
- **Dennis Rilling** (JLC)
- **John Major** (Motorola)
- **Lydia Shen** (NRAD)
- **Philip Hausler** (IBM)
- **CDR Steve Christensen** (PEO-TacAir)
- **Stu Rider** (Mobil Oil)
- **Lock Yung** (Army/PEO)
- **William Wilder** (NAVSEA)
- **Dennis Ahern** (Westinghouse)
- **Richard Mitchell** (NAWC)
- **Gary Sundberg** (Lockheed Martin/Colorado Springs)
- **Tom Gilb** (Author/Norway)
- **Terry Gill** (CMRI)
- **Rick Berthiume** (TASC)
- **Axel Ahlberg** (General Electric)
- **Dick Dye** (CTA)
- **Danny Holtzman** (Vanguard Research)
- **Pat Pierce** (SAIC)
- **Ken Murphy** (Rational)
- **Dan DeJohn** (Digicomp Research Corp)
- **David Weisman** (Unisys)

Process Improvement

- **MAJ Paul Zappala** (MCTSSA)
- **Jack Fergason** (SEI)
- **Mary Ellen Claget** (NSA)
- **Beth Springstein** (IDA)
- **MAJ George Newberry** (AFSAF/AQ)
- **Tom Goodall** (NSWC/PHD/ECO)
- **Gary Christle** (OUSD (A&T)/API)
- **Don Reifer** (DISA)
- **Gary Petersen** (AF-STSC)
- **Dave Cashpular** (Army)
- **Lyn Dellinger** (DSMC)
- **Jim Dobbins** (DSMC)
- **Rob LeiBrandt** (DAU)
- **Christine Davis** (TI)
- **Michael Condry** (Sun)
- **Lewis Gray** (Ada Pros Inc.)
- **Larry Migdahex** (Keane Fed. Sys)
- **Rich Lordahl** (Unisys)
- **Jim Chelini** (Raytheon)
- **Dave Whitten** (TI)
- **Ken Schumate** (Hughes)
- **James Collofello** (Arizona State Univ.)
- **Lloyd Anderson** (DISA/Honeywell/ICASE)
- **Jack Kramer** (IDA)
- **Arthur Pyster** (Software Productivity Consortium)

Solicitation & Contracting

- **Elliott Branch** (OASN/RDA)
- **COLRichard Heffner** (SAF/AQCI)
- **Bob Schwenk** (Army/Dir Info Sys)
- **Joe Sousa** (Navy/ASN (RDA) (APIA))
- **Al Selgas** (JLC)
- **LT Charles Race** (Navy PG School)
- **Ron Larson** (PEO-CU)
- **Raj Avula** (PEO-TAD)
- **CAPT Bob McArthur** (Marine Corps-MCTSSA)
- **Bill Mounts** (OUSD (A&T) (ARA))
- **Bob Finkelman**
- **Mike Dyer** (Lockheed Martin)
- **David Maibor** (David Maibor Assoc.)
- **Bob Mellott**
- **Bob Mutchler** (General Research Corp)
- **Sam Davis** (Lockheed Martin)
- **Art Buchannan** (Mitre)
- **Jim Gottfried** (SAIC)
- **Jack Allahand** (Lockheed Martin)
- **Mike Bennett** (Logicon)

BEST PRACTICES INITIATIVE CONTRIBUTORS

THE PROGRAM MANAGERS PANEL

Dan Fisher	Rational
Kathy Hegmann	Loral/Manassas
Bob Knickerbocker	Lockheed Martin Corporation
Ron Morrison	Hughes
Al Whittaker	Lockheed Martin Corporation

THE AIRLIE SOFTWARE COUNCIL

Victor Basili	University of Maryland
Grady Booch	Rational
Norm Brown	Software Program Managers Network
Peter Chen	Chen & Associates, Inc.
Christine Davis	Texas Instruments
Tom DeMarco	The Atlantic Systems Guild
Mike Dyer	Lockheed Martin Corporation
Mike Evans	Computers & Concepts Associates
Bill Hetzel	Qware
Capers Jones	Software Productivity Research, Inc.
Tim Lister	The Atlantic Systems Guild
John Manzo	3Com
Lou Mazzucchelli	Gerard Klauer Mattison & Co., Inc.
Tom McCabe	McCabe & Associates
Frank McGrath	Software Focus, Inc.
Roger Pressman	R.S. Pressman & Associates, Inc.
Larry Putnam	Quantitative Software Management
Howard Rubin	Hunter College, CUNY
Ed Yourdon	American Programmer

BIBLIOGRAPHY

BIBLIOGRAPHY

BIBLIOGRAPHY

Air Force Materiel Command Pamphlet (AFMCP) *Software Development Capability Evaluations*, June 1994. Defines a methodology for assessing an organization's capability to develop software for mission-critical computer systems.

L.J. Arthur, *Rapid Evolutionary Development*, John Wiley & Sons, Inc., New York, 1992.

P. Bell, *Fast, Furious, Frenzied, and Fun*, Pfeiffer & Company, San Diego, 1994. Advice and templates for starting up and controlling new departments.

P. Bell and C. Evans, *Mastering Documentation*, John Wiley & Sons, Inc., New York, 1989. Templates for project and product planning and progress documentation.

B. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, Reading, Massachusetts, 1975. A classic on software measurement and control problems by a pioneer in mainframes and virtual reality.

F. P. Brooks, "No Silver Bullet: Essence & Accidence of Software Engineering," *IEEE Computer*, Vol. 20, No. 4, April 1987.

R. X. Cringely, *Accidental Empires*, Addison-Wesley Publishing Company, Inc., New York, 1992. How computer companies, especially Apple, work, and don't.

T. DeMarco and T. Lister, *Peopleware*, Dorset House, 1987.

Directions for Defense: Report of the Commission on Roles and Missions of the Armed Forces, May 24, 1995.

P. Drucker, *Managing for the Future*, Penguin Books, 1992.

K. Ermel, P. Perry, J. Shields, *Insiders Guide to Software Development*, Que Corporation, 1994.

M. Evans, *The Software Factory*, John Wiley & Sons, Inc., 1989. The crafts and approaches involved in managing software development.

M. Evans, *Principles of Productive Software Management*, John Wiley & Sons, 1983. Basic principles for managing software systems.

T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley Publishing Company, 1988. Evolutionary design steps.

G. Gilder, *Microcosm*, Simon & Schuster, Inc., 1989.

J. Gleick, *Chaos: Making a New Science*, Viking, New York, 1987. A theory that produces not only great calendar art but a new way to understand dynamic processes.

R. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, 1993. Practical uses of metrics within software projects.

P. Hall, *Great Planning Disasters*, University of California Press, Berkeley, 1980.

W. Humphrey, *Managing the Software Process*, Addison-Wesley, 1989. A fundamental book for software project management.

C. Jones, *Applied Software Measurement*, McGraw-Hill, 1991. A fundamental book on measurement.

C. Jones, *Assessment and Control of Software Risks*, Yourdon Press, 1994.

C. Jones, *Software Measurements of Best-In-Class Organizations (Draft for Application Development Trends)*, 1994. Measures that the best software producers are trying to achieve.

J. Kawanami, "Re-engineering the Enterprise," *Data Management Review*, March 1995. Workbenches and the 80/20 development approach.

R.E. Kraut and L.A. Streeter, "Coordination in Software Development," *Communications of the ACM*, Vol. 38, No. 3, March 1995. Handling the inevitable problems in software development projects.

B. Laurel, ed., *The Art of Human-Computer Interface Design*, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1990.

S. McGuire, *Debugging the Development Process*, Microsoft Press, 1994. Up-close look at development projects in the world's biggest software company.

M. Norris, P. Rigby, and M. Payne, *The Healthy Software Project: A Guide to Successful Development and Management*, John Wiley & Sons, 1993. An easy-to-read, practical guide to identifying the status of software projects.

F. O'Connell, *How to Run Successful Projects*, Prentice Hall, 1994. An easy-to-read guide to managing software projects.

J. Palfreman and D. Swade, *The Dream Machine*, BBC Books, London, 1991.

P. Perry and K. Ermel, *Insider's Guide to Software Development*, Que Corporation, 1994. How software gets developed, and doesn't, in the commercial world.

W. Poundstone, *Prisoner's Dilemma*, Doubleday, New York, 1992. A quick, easy way to illustrate the value of teamwork to everyone in an organization.

R. Pressman, *A Manager's Guide to Software Engineering*, McGraw-Hill, 1993. An extensive reference.

L. Putnam, *Measures for Excellence*, *Reliable Software on Time and Within Budget*, Yourdon Press, 1992. An extensive model for software estimation.

D. Taylor, *Object-Oriented Technology: A Manager's Guide*, Servio Corporation, Alameda, California. A short, business-oriented book for program managers who need to understand the basic benefits and risks of object orientation.

R. Thomsett, *Third Wave Project Management*, Yourdon Press Computing Series, 1993. A handbook for managing complex information systems in the 1990s.

BIBLIOGRAPHY

G.M. Weinberg, *Quality Software Management, Vol. 2, First Order Measurement*, Dorset House Publishing, 1993.

F. Wellman, *Software Costing*, Prentice Hall, 1992.

E. Yourdon, *Decline and Fall of the American Programmer*, Yourdon Press, 1993.

E. Yourdon, *Guerrilla Programmer*, Vol. 2, No. 2, February 1995. Industry news of interest.

INDEX

INDEX

Q

R

S

T

U

V

W